Week 1 - Practical aspects of Deep Learning

07 June 2020 16:06

Setting up your Machine Learning Application.

TRAIN / DEV / TEST Sets

Layers
Hidden units
Activation fx
Learning rates ....

Predict good parameters.

data = [Train (60%) | dev / CV / Cross Validation Set (20%) | Test (20%)]

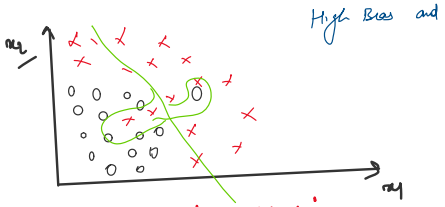
Not having a test set is okay and only have Dev Set;

BIAS / VARIANCE

Train set Error :→ 1%
Dev set Error :→ 11%

✓ Overfitting / High Variance
✗

High Bias and High Variance



Basic Recipe for Machine Learning

(1) High Bias ?? (Training data performance) ——Y——→
- Bigger N/w Layer
- Layer optimization Algos

(2) High Variance ? ——→ Y ——→ (Dev set performance)
- More data
- Regularization
- Neural N/w Architecture

Done!

" bias / Variance Trade off

bias ↑ Variance ↓

Regularizing your Neural N/w

$$\|w^{[l]}\|^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} \left(w_{ij}^{[l]}\right)^2$$

Regularization

Logistic Regression —

$$\min_{(w,b)} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2}\|w\|^2$$

$w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

L2 Regularization: $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

L1 Regularization: $\frac{\lambda}{m}\|w\|_1$ (λ = Regularization parameter)

L2 Regularization → $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

L1 Regularization → $\frac{\lambda}{m} \sum_{i=1}^{n_x} |w| = \frac{\lambda}{m} \|w\|_1$

(λ = Regularization parameter)
[Hyperparameter]

Neural N/w

$J(w^{[1]}, b^{[1]}, \ldots) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \|w^{[l]}\|^2$

$\|w^{[l]}\|^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{ij}^{[l]})^2$ } "Frobenius norm" $\|\cdot\|_F^2$

$w \in (n^{[l]}, n^{[l-1]})$

$dw = (\text{Calculated from Backprop}) + \frac{\lambda}{m} w^{[l]}$

$\frac{\partial J}{\partial w^{[l]}} = dw^{[l]}$

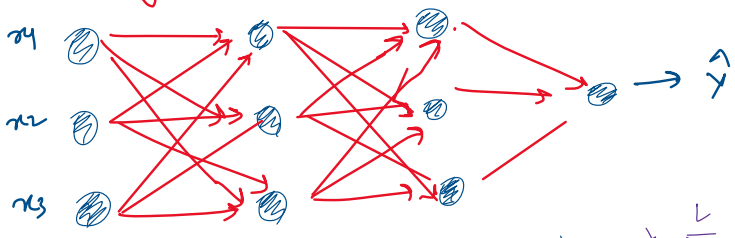
$w^{[l]} = w^{[l]} - \alpha\, dw^{[l]}$

\# L2 regularization is also called "weight decay"

$w^{[l]} := w^{[l]} - \alpha \left[(\text{Backprop}) + \frac{\lambda}{m} w^{[l]} \right]$

$= w^{[l]} - \frac{\alpha\lambda}{m} w^{[l]} - \alpha [(\text{Backprop})]$

Why Regularization prevents Overfitting ?

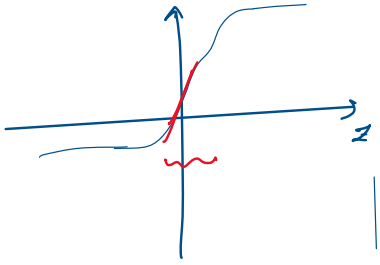


$J(w^{[l]}, b^{[l]}) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \|W^{[l]}\|_F^2$

⊙ (If λ » big, $w \approx 0$ almost)

⊙ high bias → high variance

tanh(fx)



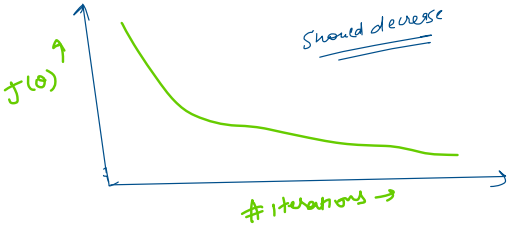
$g(z) = \tanh(z)$

$w^{[l]} \downarrow$, $\lambda \uparrow$

$f(x)$ will be linear in red range

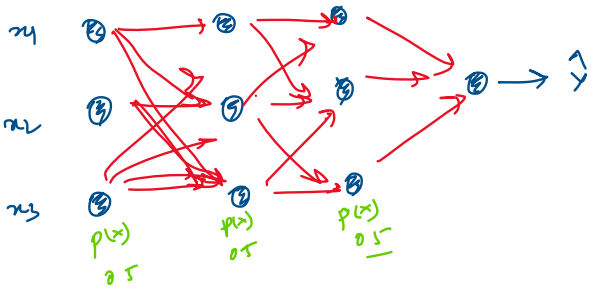
\# As if every layer is linear N/w

$J(\,) = \sum \mathcal{L}(\hat{y}, y) + \frac{\lambda}{2m} \sum_{l} \|W^{[l]}\|_F^2$



DROPOUT REGULARIZATION

(#) Set some probability to dropout a node in n/w,



Implementing Dropout (Inverted dropout)

illustrate in layer $l=3$:

$d_3 = np.random.rand(a_3.shape[0], a_3.shape[1]) < keep\_prob$

keep_prob = 0.8 (say)

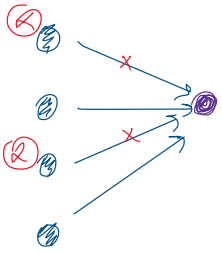
$a_3 = np.multiply(a_3, d_3)$ ($a_3 *= d_3$)

$a_3 /= keep\_prob$ (Inverted dropout technique)

(#) (don't dropout at test time.)

Understanding Dropout

(#) Can't rely on one feature, so have to spread weights
≈ Shrink weights



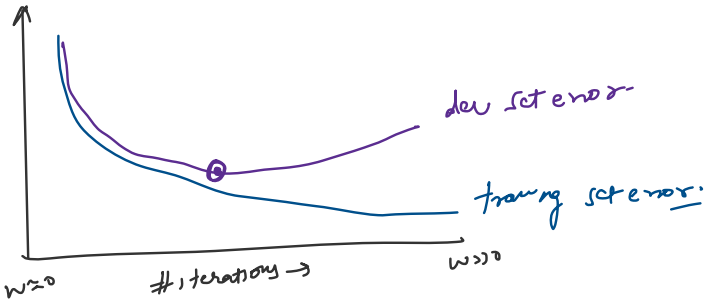
(i) Can vary keep prop according to layer,

units $\propto \frac{1}{keep\ prop\ value}$

(#) Computer vision

Other Regularization Methods

EARLY STOPPING



optimize cost fx J (Gradient descent, LBFGS, ...)
do not overfit
(Regularization, getting more data)

→ Orthogonalization → Thinking of one task at a time.

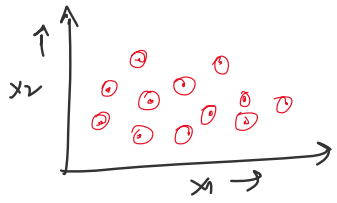
→ Orthogonalization → Thinking of one task at a time.

→ (-) No orthogonalization in early stopping

L_2 Regularization

Setting up your Neural N/W

Normalizing Inputs



$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

① Subtract Mean

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$x := (x - \mu)$$

② Normalize Variances

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}\left[x^{(i)} \ast\ast 2\right] \quad \text{(Squaring)}$$

$$x = x/\sigma$$

$$\boxed{X = (x-\mu)/\sigma}$$

(-) use same μ and σ^2 to normalize Test set.

WHY NORMALIZE ?

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

w_1 - 1 - 1000

w_2 - 1 - 10

Vanishing / Exploding gradients

$g(z) = z$

$\hat{y} = W^{[L]} * W^{[L-1]} * W^{[L-2]} * W^{[L-3]} * \cdots$

$W^{[3]}\ W^{[2]}\ W^{[1]}(X)$

$z^{(1)}$

$a^{(1)} = g(z^{(1)})$

$a^{(1)} = z^{1}$

$z^{(1)}$

$z^{[3]}$

$W^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$ (1.5 | 0.5)

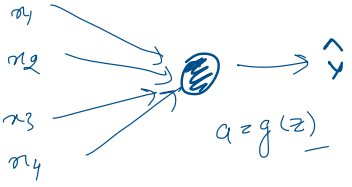
$\hat{y} = W^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} X$

$\hat{y} = (1.5)^L$

* 1.5 times number of layers
* If you have deep neural n/w value of $\hat{y}$ will explode.

WEIGHT INITIALIZATION FOR NEURAL N/W.

Single Neuron



$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$

$b = 0$

$n \propto \frac{1}{w^i}$

$Var(w^i) = \frac{1}{n}$ [$n \in$ No of input features in neuron

$W^{[l]} = np.random.randint(shape) * np.sqrt\left(\frac{1}{n^{[l-1]}}\right)$

$2/n$ for RELU (fx.)

$g^{(l)}(z) \in$ Relu

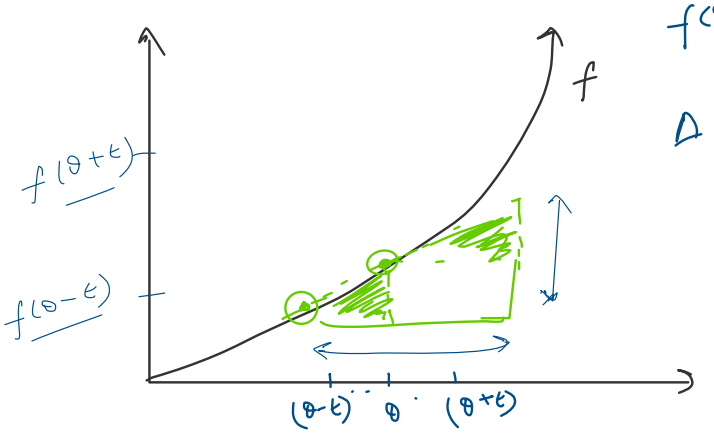
Other Variants

$\tanh(z) = \sqrt{1/n^{[l-1]}}$ { Xavier initialization. }

NUMERICAL APPROXIMATION OF GRADIENTS

Backprop (Gradient checking).

$f(\theta) = \theta^3$



$f(\theta) = \theta^3$

$$\Delta = \lim_{\epsilon \to 0} \frac{f(\theta+\epsilon) - f(\theta-\epsilon)}{2\epsilon}$$

$\approx g(\theta)$

GRADIENT CHECKING.

gradient checking for Neural N/W :→

Take $W^{[1]}, b^{[1]} \dots\dots W^{[L]} B^{[L]}$ & reshape to a big vector, θ.

(concatenate)

take $dW^{[1]}\ db^{[1]}, dW^{[2]}\ db^{[2]} \dots\dots dW^{[L]}\ db^{[L]}$ & reshape into a big vector, $d\theta$

(concatenate →)

grad check

for each i :→

$$d\theta_{approx}^{(i)} := \frac{J(\theta_1, \theta_2, \theta_3 \dots \theta_i + \epsilon) - J(\theta_1, \theta_2, \theta_3 \dots \theta_i - \epsilon)}{2\epsilon}$$

$\approx d\theta(i)$

$= \frac{\partial J}{\partial \theta_i}$

$$d\theta_{approx} \approx\approx d\theta$$

Check $= \frac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \| + \| d\theta \|_2}$

$\epsilon \approx 10^{-7}$

Gradient Checking Implementation

(*) dont use grad check in training only to debug.

(*) if algorithm fails in grad check, look at the components & try to find the bug.

$db^{[l]}$, $dw^{[l]}$

(*) Remember Regularization

$$J(\theta) = \frac{1}{m} \sum \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l} \| w^{[l]} \|_2^2$$

$d\theta$ = grad of J wrt θ

(*) grad check doesn't work with dropout

(*) run at random initialization; perhaps after some training.

Week 2- Optimization Algorithms

08 June 2020 23:43

Mini batch gradient descent.

$X = [x^{(1)}, x^{(2)}, x^{(3)} \ldots x^{(m)}]$

(n_x, m)

$Y = [y^{(1)}, y^{(2)}, y^{(3)} \ldots y^{(m)}]$

$(1, m)$

if training examples are very large, say 10 million.

(#) will have to traverse (10 million) training set before you take one little step of gradient descent.

→ Mini-Batches

Now, One mini batch consists of = $\frac{\text{training-examples}}{\text{n. of mini-batches}} = n(mb)$.

if $X = 50$ (say) & $n = 5$

$X^{\{1\}} = [x^{(1)} \; x^{(2)} \; x^{(3)} \; x^{(4)} \ldots x^{(10)}]$

$X^{\{2\}} = \ldots$

→ and same for Y.

Mini-Batch t : $X^{\{t\}}, Y^{\{t\}}$

⊙ $X_{dims} = n_x, n(mb)$

⊙ $Y_{dims} = 1, n(mb)$

Mini-Batch gradient descent.

for $t = 1 \ldots n(mb)$:

// perform a step of gradient descent using $X^{\{t\}}, Y^{\{t\}}$

(1) Perform forward prop. on

$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$\vdots$

$A^{[l]} = g^{[l]}(Z^{[l]})$

(2) Compute cost

$J = \frac{1}{m} \sum \mathcal{L}(\;) + \text{Regularization Term}$

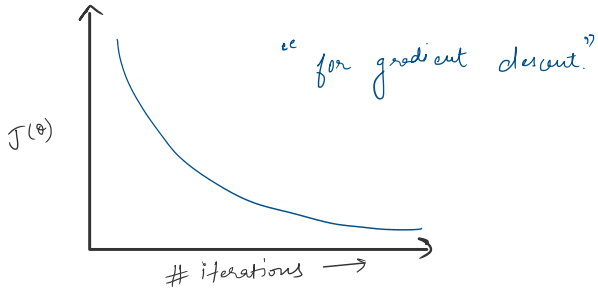
$W^{[l]} := W^{[l]} - \alpha \, dW^{[l]}$ & similarly,

$b^{[l]} := b^{[l]} - \alpha \, db^{[l]}$

" Also called 1 epoch / 1 pass

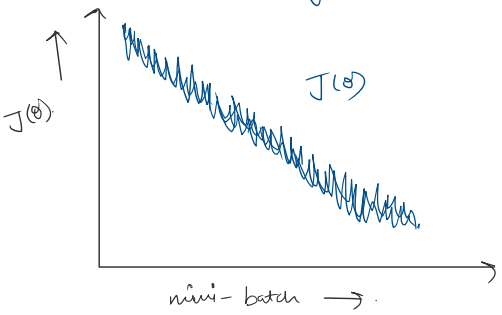
L2. UNDERSTANDING MINI BATCH GRADIENT DESCENT.

12. Understanding Mini Batch Gradient Descent.



"for gradient descent."

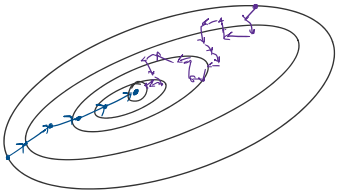
Mini – Batch gradient descent



Choosing size of mini batch !

* if mini batch size = m, then : Batch gradient descent.
* if (mini batch size = 1) : Stochastic gradient descent.

$(x^{\{1\}}, y^{\{1\}}, x^{\{2\}}, y^{\{2\}}, \ldots\ldots)$



[In practice mini batch size is between 1 and m.

Batch gradient descent	Stochastic gradient descent
mini - batch size : m	batch size : 1
Too long per iteration	loose speedup from vectorization.

→ mini - batch gradient descent

Choosing size of mini - batch

if Small training set : Use batch gradient descent.

if Small training set : Use batch gradient descent.
$(m \leq 2000)$

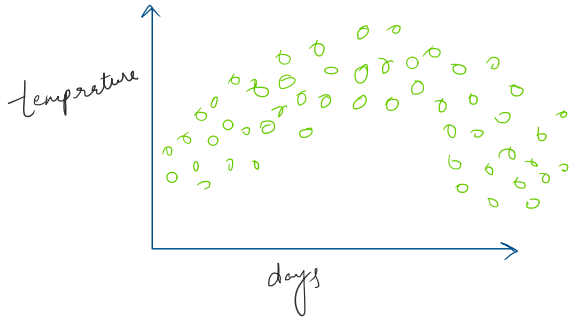
Typical mini-batch sizes : 2^n (multiple of 2.)

Make Sure mini-batch fits in CPU | GPU memory.
$X^{\{t\}}, Y^{\{t\}}$ → Hyperparameter.

13. EXPONENTIALLY WEIGHTED AVERAGE

Temperature in New-delhi :→

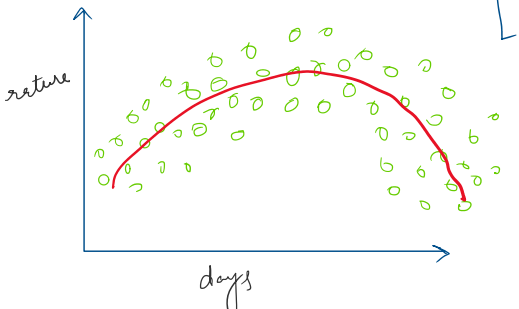
$\theta_1 = 32^\circ C$
$\theta_2 = 31^\circ C$
$\theta_3 = 29^\circ C$
$\theta_{180} = 28^\circ C$



$V_0 = 0$
$V_1 = 0.9 V_0 + 0.1 \theta_1$
$V_2 = 0.9 V_1 + 0.1 \theta_2$
$V_3 = 0.9 V_2 + 0.1 \theta_3$

$$V_t = 0.9 V_{(t-1)} + 0.1 \theta_t$$

→ Exponentially weighted average.
→ Moving Average.



$$V_t = \beta V_{(t-1)} + (1-\beta) C$$

Exponentially weighted moving average.

→ generalize formula.

$\beta = 0.9$

$V_t \overset{(avg)}{\approx} \frac{1}{(1-\beta)}$ days temperature.

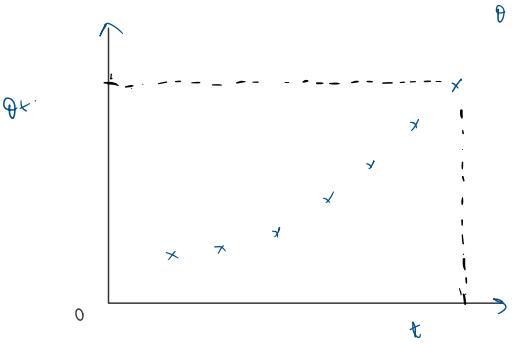
L4. Understanding Exponentially Weighted Avg.

$V_t = \beta V_{(t-1)} + (1-\beta)\theta_t$

$V_{100} = 0.1\theta_{100} + 0.9V_{99}$ $\quad \beta = 0.9$

$= 0.1\theta_{100} + 0.1 * 0.9(\theta_{99}) + 0.1(0.9)^2\theta_{98} + \ldots$

- exponentially decaying function.



(exp. decaying fx.)

$0.9^{10} \simeq 0.35 \simeq \frac{1}{e}$

$(1-\epsilon)^{1/\epsilon} = \frac{1}{\epsilon}$

Avg. over 10 days.

- $V_0 = 0$
- $V_t = \beta V_{(t-1)} + (1-\beta)\theta_1$

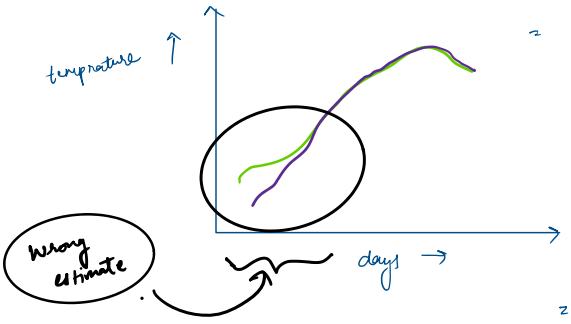
$V = 0$

$V_0 = \beta V + (1-\beta)\theta_1$

$V_0 = \beta V + (1-\beta)\theta_2$

L5. Bias Correction in Exp. Weighted Avg.

$V_t = \beta V_{t-1} + (1-\beta)\theta_t$; $\boxed{\beta = 0.98}$



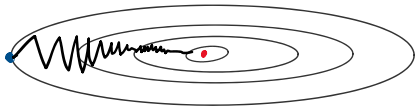
$= \frac{V_t}{(1-\beta^t)}$

$\boxed{t=2}$:

$= \frac{V_2}{1-\beta^2} = \frac{}{(1-0.98)^2}$

$= \frac{V_2}{0.0396}$

L6. Gradient Descent with Momentum



↕ — Slower learning.

↔ — faster learning.

Momentum :-

On every iteration t, Calculate :-

dw, db on current batch → friction

$V_{dw} = \beta V_{dw} + (1-\beta)dw$ ← Acceleration

weights

dw, db on current batch → fraction

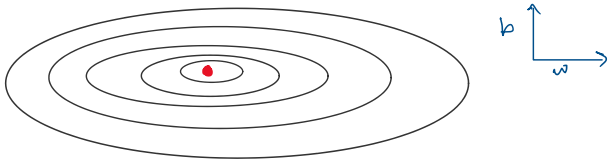
Velocity ← $V_{dw} = \beta V_{dw} + (1-\beta) dw$ } — Acceleration
$V_{db} = \beta V_{db} + (1-\beta) db$

$w := w - \alpha V_{dw}$

$b := b - \alpha V_{db}$

Hyperparameters :- α, β $\quad \boxed{\beta = 0.9}$

L7. RMS PROP (Root mean square prop)



On every iteration, t :-

Compute dw, db for mini-batch.

$S_{dw} = \beta S_{dw} + (1-\beta_2) dw^2$ — element wise square.

β is hyperparam for momentum

$S_{db} = \beta S_{db} + (1-\beta_2) db^2$

$S_{dw} \approx 0$

to make sure it's +ve, add a small value ϵ.

$\epsilon \approx 10^{-8}$

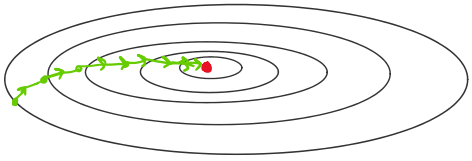
$w := w - \alpha \frac{dw}{\sqrt{S_{dw}} + \epsilon}$

$b := b - \alpha \frac{db}{\sqrt{S_{db}} + \epsilon}$

(*) $w \rightarrow$: fast.
$b \uparrow$: slow down.

S_{dw} = relatively small.

S_{db} = relatively large.



* Rms prop.

→ now, can also use large learning rate, α.

L8. ADAM's OPTMIZATION ALGORITHM

{ gd (Momentum) + rmsprop. }

$V_{dw} = 0$, $S_{dw} = 0$; $V_{db} = 0$ $S_{db} = 0$

On every iteration t :

* Compute dw, db using current mini-batch.
* $V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw$
  $V_{db} = \beta_1 V_{db} + (1-\beta_1) db$
  } • Momentum (Hp: β_1)
* $S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$
  $S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$
  } • rms prop. (Hp: β_2)

Bias correction

* $V_{dw}^{corrected} = V_{dw} / (1-\beta_1^t)$
  $V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$ } • Momentum

$S_{dw}^{corrected} = V_{\partial b} / (1-\beta_1^t)$
$S_{db}^{corrected} = V_{\partial b} / (1-\beta_1^t)$ } • Rms prop.

* $w := w - \alpha \left[\frac{V_{dw}^{corrected}}{\left(\sqrt{S_{dw}^{corrected}} + \epsilon\right)} \right]$

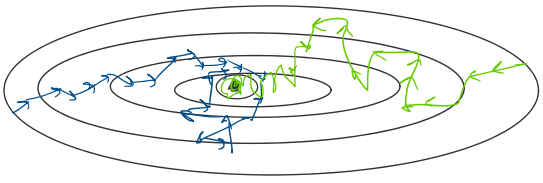
* $b := b - \alpha \left[\frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \epsilon} \right]$

Hyperparameters choice :-

α = needs to be tuned.
β_1 = 0.9 (dw)
β_2 = 0.999 (∂w^2, ∂b^2)
ϵ = 10^{-8}

ADAM : "Adaptive moment Estimation."

L9. LEARNING RATE DECAY :



Slowly reduce, α

* 1 epoch = 1 pass over the data.

$\alpha = \left(\frac{1}{1 + \text{decay rate} * \text{epoch num}} \right) \alpha_0$ → some initial learning rate.

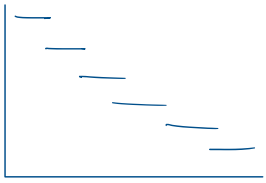
Epoch	α
1	0.1
2	0.067
3	0.5
4	0.4

$\alpha_0 = 0.2$
decay = 1

$\alpha = \left[(N<1)^{\text{epochnum.}} \cdot \alpha_0 \right]$:- exponential decay.

people also use:

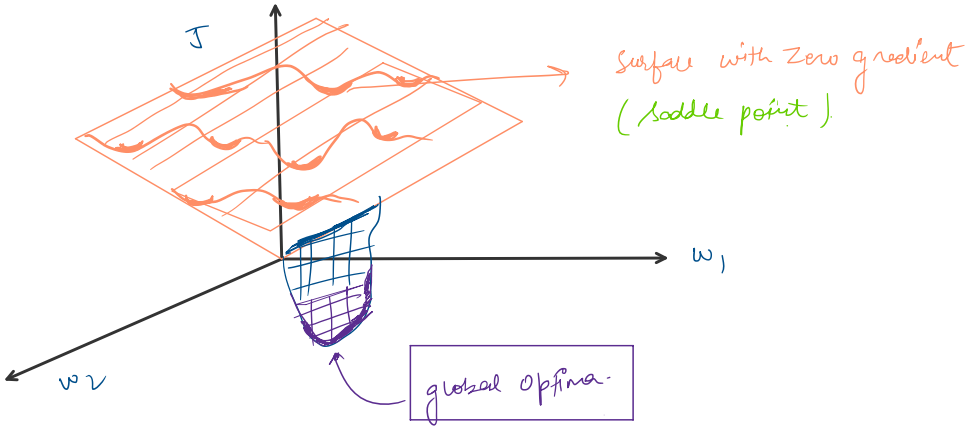
$\alpha = \frac{k}{\sqrt{\text{epochn.}}} * \alpha_0$



:- descrete staircase.

* Manual decay.

L10: The Problem Of Local Optima



Problem with plateaus.

* [plateau is a region where the derivative is 0 for a very long time.

- Unlikely to stuck in a bad local optima.
- Plateaus can make learning slower.

Week 3 - Hyperparameter tuning, Batch Normalization and programming frameworks

11 June 2020 17:26

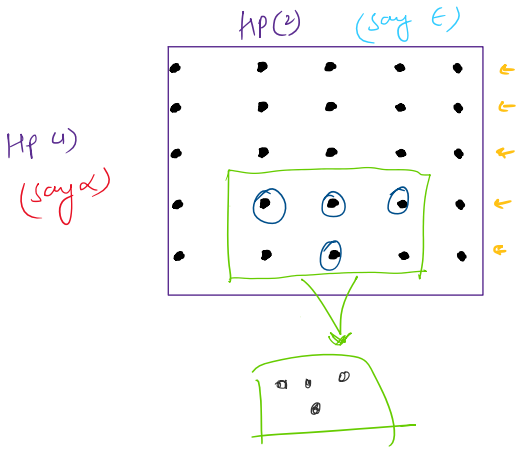
Hyperparameter Tuning

L1. Tuning process.

Hyperparameters.

α, β (~.9), β_1 (.9), β_2 (.999), ϵ (10^{-8}), #layers, #hidden units, learning rate decay, mini-batch size.

* - high priority.
* — Mediocre priority.
* — Low priority.
* — almost negligible (default values are used.)



25 values

{ Randomly pick these hp's and find that works best.

- Coarse to find sampling scheme.

* Use random Sampling and adequate search.
{optional} * Coarse to find search process.

L2. Using An Appropriate Scale To Pick HP's.

α 0.0001 ... 1

90% Values $\in [0.1, 1]$

using log scale

.0001 .001 .01 1

$a = \log_{10}(.0001) = -4$ (10^{-4})
$a = -4$
$r = -4 * np.random.rand()$

(10^0) $b = \log_{10}(1)$
$b = 1$
$r = [-4, 0]$

$a = \log_{10}(.001)$

$a = -4$

$r = -4 * np.random.rand()$

$\alpha = 10^r$

Sample r uniformly on $\in [a, b]$

$r = [-4, 0]$

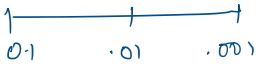
$\alpha = [10^{-4}, 10^{0}]$

Hp's for exponentially weighted avg.

$\beta = [0.9 \text{ ---- } 0.999]$

Linear Scale — ⊗

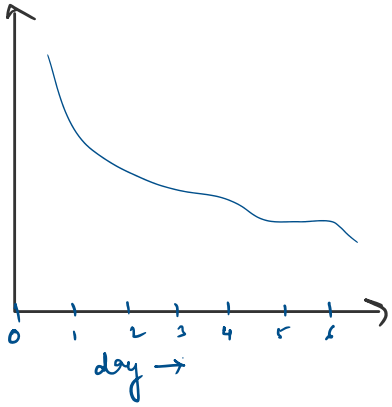
$1-\beta = [0.1 \text{ ---- } 0.001]$ — ✓



Very sensitive to small changes in β.

L3: HP tuning (Panda v/s Caviar)

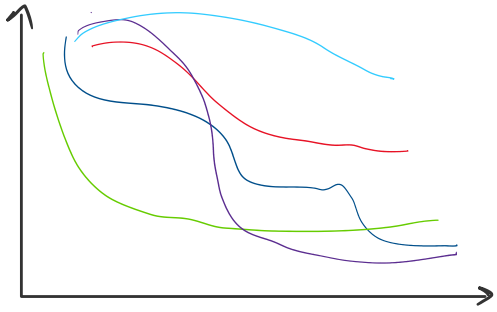
Babysitting one model.



Keep checking one day at a time.

Panda Approach.

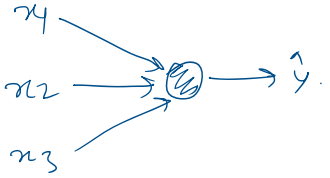
Training many models in Parallel



Caviar Approach.

Batch Normalization

L1: Normalizing Activations in a N/W



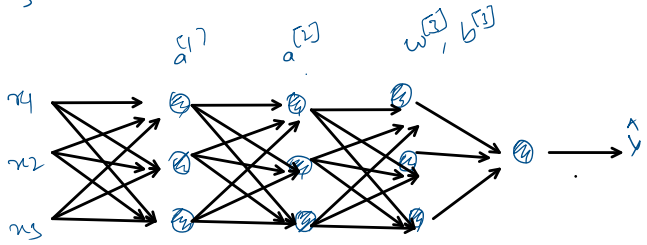
$X = X - \mu$

$\sigma^2 = \frac{1}{m} \sum_i x^{i\,2}$

$X /= \sigma^2$

$z^{[1]}$ $a^{[1]}$ $w^{[1]}, b^{[1]}$

x_3



Can we normalize $a^{[2]}$? to train $w^{[3]}$ $b^{[3]}$ faster ?

* Normalize $z^{[2]}$ or $z^{[2]}$ instead of $a^{[1]}$

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$$

$$z^{(i)}_{normalize} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

z has normal distribution and Variance 1, but we want different dist.

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

Learnable parameter.

Standardize mean and Variance.

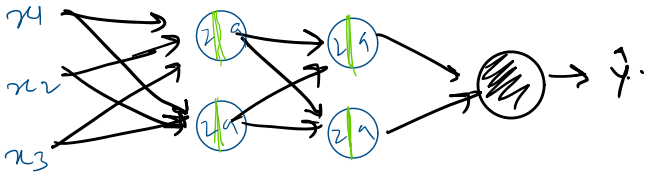
if $\gamma = \sqrt{\sigma^2 + \epsilon}$ and $\beta = \mu$ {

then :

$$\tilde{z}^{(i)} = z^{(i)}$$

Use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$

L2. Fitting Batch Norm to a N/W :>



$$X \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[(B \cdot N)]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} = g(\tilde{z}^{[1]}) \rightarrow z^{[2]} \cdots\cdots$$

Parameters :> $w^{[1]}, b^{[1]}, \ldots, \beta^{[1]}, \gamma^{[1]}$

$$\beta^{[1]} := \beta^{[1]} - \alpha \cdot d\beta^{[1]}$$

- **Working with mini-batches.**

$$X^{\{1\}} \xrightarrow{w^{[1]}, b^{[1]}} \tilde{z}^{[1]} \xrightarrow[(B \cdot N)]{\beta, \gamma} g^{[1]}(\tilde{z}^{[1]}) \rightarrow \cdots\cdots$$

$g^{[1]}(\tilde{z}^{[1]}) = a$

$w^{[l]}, b^{[l]}$ → $(B \cdot N)$ → a

Parameters = $w^{[l]}, b^{[l]}, \beta^{[l]}, \gamma^{[l]}$ ⇒ $z^{[l]} = w^{[l]} * a^{[l-1]} + b^{[l]}$ (× ×)

* Batch Norm zeroes out all $z^{[l]}$, so adding/subtracting any any const. doesn't effect the equation.

$$z^{[l]} := w^{[l]} a^{[l-1]}$$

$$z^{[l]}_{norm} := \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{z}^{[l]} = \gamma^{[l]} z^{[l]} + \beta^{[l]}$$

dims :- $(n^{[l]}, 1)$

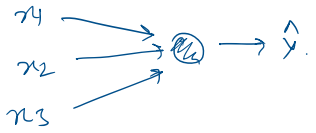
Implementing gradient descent

for t=1 to ... Mini-Batch(n) :-

- Compute forward prop on $X^{\{t\}}$
- use batch Norm to replace $z^{[l]} \rightleftharpoons \tilde{z}^{[l]}$
- use backprop to calculate grads.
- update parameters, → $w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$
  → $\beta^{[l]} := \beta^{[l]} - \alpha\, d\beta^{[l]}$
  → $\gamma^{[l]} := \gamma^{[l]} - \alpha\, d\gamma^{[l]}$

L3. Why does batch norm works?

* learning on shifting input dist.



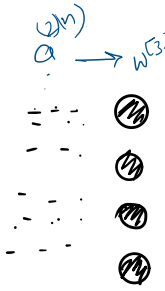
Trained on monochrome image. Trained on coloured image

— "Covariate Shift" →

* If you learn some $x \to y$ mapping, if distribution of x changes then you need to retrain your learning algorithm.

→ Covariate Shift in N.N

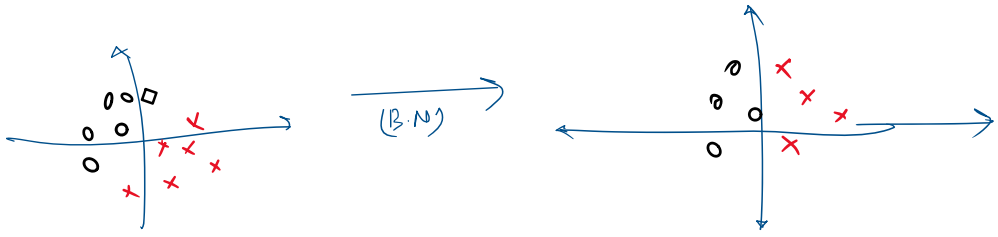
→ Covariate Shift in N.N



Problem

→ parameters w, b before this layer change, $a^{[2]}$ will also change. Suffering from a problem of "Covariate Shift".

"Batch norm : No matter the distribution changes, mean and variance remains same.



Batch Norm as regularization.

- Each mini-batch is scaled by mean/variance computed on just one batch.
- This adds some noise $z^{(i)}$, scaling process of $z^{[i]} \to \tilde{z}^{[i]}$ becomes more noisier, similar to dropout.
- This has slight regularization effect.

mini-batch size $\propto \frac{1}{\text{noise}} \propto$ regularization.

(*) "don't use batch norm as regularization only use for normalization." ✗ ✓

L4. Batch Norm at test time.

We need different μ and σ^2.

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

[m :- ex in mini-batch.

$$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$$

$$\to z^{(i)} = z^{(i)} - \mu$$

$$Z_{norm}^{(i)} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{Z}^{(i)} = \gamma Z_{norm} + \beta$$

estimate using exponentially weighted avg. (across mini-batch)

$X^{\{1\}} \to \mu^{\{1\}[1]}$ (θ_1) $\sigma^{2\{1\}[1]}$

$X^{[2]} \to \mu^{\{2\}[1]}$ (θ_2) $\sigma^{2\{2\}[1]}$

$X^{[3]} \to \mu^{\{3\}[1]}$ (θ_3) $\sigma^{2\{3\}[1]}$

---

new Z_{norm}

$$Z_{norm} = \frac{Z - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad \tilde{Z} = \gamma Z_{norm} + \beta$$

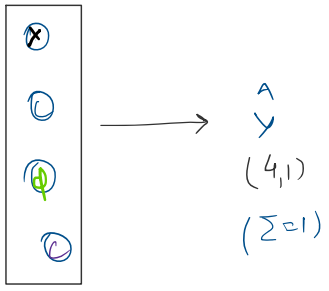
Multi Class Classification

4. Softmax Regression.

Now recognize cats, dogs and chicks.

Class1 Class2 Class3 -1.

classes : 4



$\hat{y}$

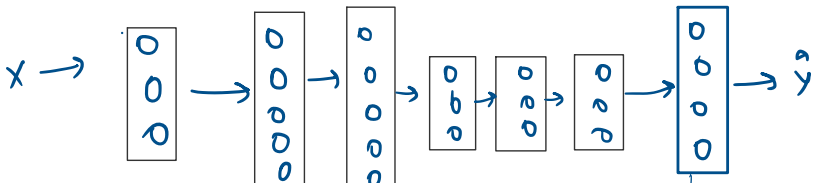
$(4,1)$

$(\Sigma = 1)$

— final layer —

Softmax Layer

Softmax Layer



$$z^{[L]} = W^{[L]} * a^{[L-1]} + b^{[L]}$$

Softmax Activation fx

$$t = e^{(z^{[L]})}$$

$(4,1)$

$$a^{[L]} = \frac{e^{(z^{[L]})}}{\sum_{i=1}^{4} t_i}, \quad a_i^{[L]} = \frac{t_i}{\sum_{i=1}^{4} t_i}$$

$(4,1)$

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \qquad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20 \end{bmatrix}$$

$$\sum_{i=1}^{4} t_i = 176.3$$

$$a^{[L]} = \frac{t}{176.3} = \frac{1}{176.3} * \begin{bmatrix} e^5 \\ e^2 \\ e^1 \\ e^3 \end{bmatrix}$$

$$a^{[L]} = g^{[L]}(z^{[L]})$$

(4×1)

L2: Training a Softmax Classifier.

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}, \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

$$a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5 / \sum t_i \\ e^2 / \sum t_i \\ e^1 / \sum t_i \\ e^3 / \sum t_i \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

"hard max"

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \} 0 \rightarrow \text{class with highest probability.}$$

* Softmax regression generalizes Logistic Regression to C classes.
  If $C = 2$, Softmax Regression = Logistic Regression.

Loss fx.

$y_{(4\times1)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ → Cat. (ground truth.)

$\hat{y}_{(4\times1)} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{C} y_j (\log \hat{y}_j)$$

$(y_1 = y_3 = y_4 = 0)$: 3 values in $\mathcal{L}(\hat{y}, y) = 0$

$$\mathcal{L}(\hat{y}, y) = -y_2 \log \hat{y}_2$$

(minimize)

$y_2 = 1 \Rightarrow -\log \hat{y}_2$

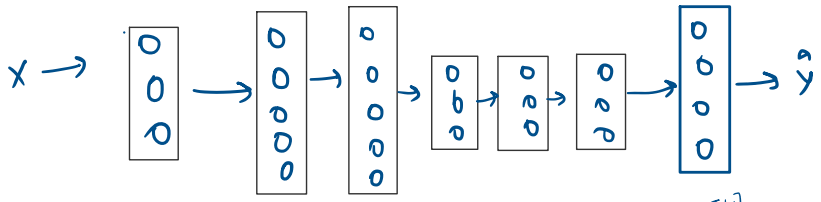
make $\hat{y}$ as big

$$J(w^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$Y = [y^{(1)} \; y^{(2)} \; y^{(3)} \cdots y^{(m)}]_{(4\times m)}$$

$$\hat{Y} = [\hat{y}^{(1)}, \hat{y}^{(2)}, \hat{y}^{(3)} \cdots \hat{y}^{(m)}]_{(4\times m)}$$

Gradient Descent with Softmax.



$z^{[l]} \rightarrow a^{[l]} \rightarrow \mathcal{L}(-)$.

$(\hat{y})$

forward prop. : ✓

backward prop. :-

$dz^{[l]} = \hat{y} - y$. $(\in 4,1)$ $(C,1)$

$\frac{dJ}{dz^{[l]}}$

Deep Learning Frameworks

- Caffe / Caffe 2
- CNTK
- DL4J
- Keras.
- Lasagne
- mxnet
- Paddle Paddle
- → Tensorflow.
- Theano
- Pytorch.

Week 1 - ML strategy (1)

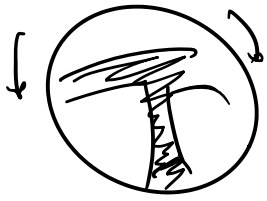
13 June 2020 11:14

Introduction to ML strategy

L2. Orthogonalization

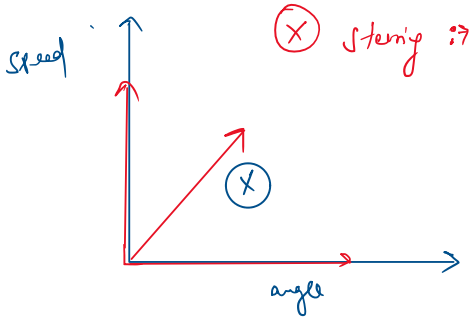
* Tuning one property at a time.

ex: driving a car



- direction : → Steering
- Acceleration, - Braking

All have separate functions to tune



(X) Steering : (0.8 * ang) + (0.3 * angle) + (0.9 * Speed)

Chain of Assumptions in ML

* fit training set well on cost function.
  - ↓
  - test set
  - ↓
  - dev set
  - ✓

Setting up your goal

L1. Single Number evaluation metric

Classifier	Precision	Recall
A	95%	90%
B	98%	85%

↑ ↑

Out of examples recognized, what % are actually cats.

What % of actual cats are correctly recognized.

What to choose?

F1-score "Avg. of Precision and recall"

$\left(\frac{2}{1/P + 1/R}\right)$ ∴ Harmonic Mean

L2. Satisficing and optimizing metrics.

Classifier	Accuracy	Running Time
A	90 %	80 ms.
B	92%	95 ms.
C	95%	1500 ms.

- maximize A
- minimize Running Time.

N metrics :- 1 optimizing metric.
(N-1) satisficing metrics.

L3. Train / dev / Test distributions.

Cat Classifier.

Regions :→ dev

- India
- Mexico
- Canada
- Spain
- Japan
- China

Test

Dev Test and test set should come from same distribution.

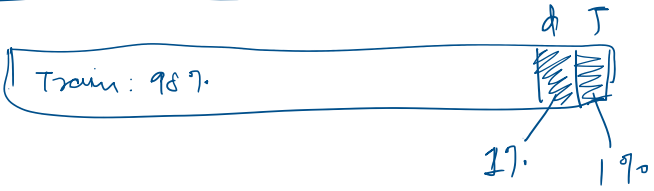
14- Size of Dev/test sets

Train :- 70% test :- 30%

or

Train :- 60% dev/CV :- 20% test :- 20%.

for 1mn, mn examples —



Size of Test Set

* Set your test size big enough to give high confidence in the overall performance of System.

15. When to change dev/test sets and metrics

Metric :- Classification error.

- ✗ Algorithm A :- 3% error → Pornographic Images
- ✓ Algorithm B :- 5% error → Ip — logs.

Misclassification.

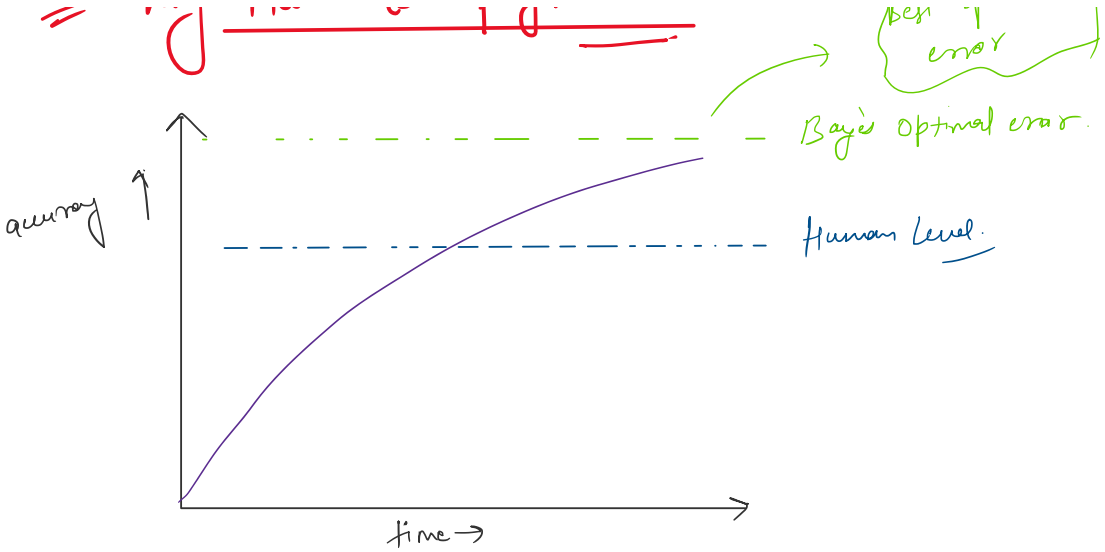
- Error :- $\frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} \{ y_{pred}^{(i)} \neq y^{(i)} \}$

$w^{(i)} = \begin{cases} 1, & \text{if related content} \\ 100, & \text{if misclassified.} \end{cases}$

Comparing to human level performance

11. Why Human level performance

Best optimal error



Avoidable Bias

Human Error :→ 1%	7.5% (say)
Training error :→ 8%	8%
dev Error :→ 10%	10%

Avoidable bias

* focus on bias

* focus on variance

Human level error is a proxy for Bayes error.

almost same content as in previous Course or Machine Learning by Andrew ng.

Week 2 - ML strategy (2)

13 June 2020 19:51

Error Analysis

11. Carrying Out error Analysis.

Cat classifier – 5% error / 95% accuracy.

- Get ~100 mislabelled dev set examples.
- Count up how many are dogs.
- Count errors.

Bias and Variance with mismatched data distribution.

Assume Human got ≃ 1% error.

Training error :> 1%

Dev Error :- 10%

Variance Error.

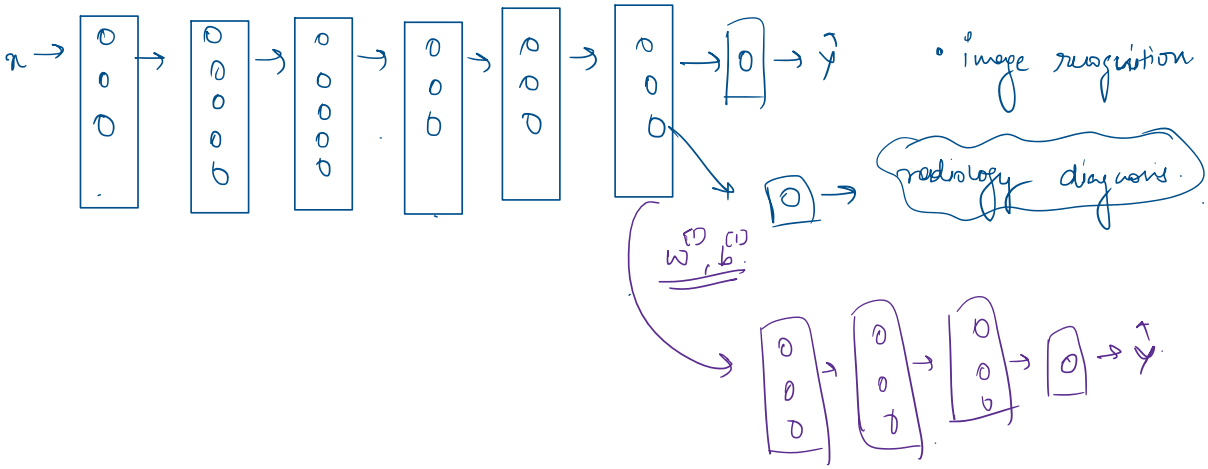
Explain almost same as train/test/dev split in 2nd course.

data mismatch problem:

- Carry out manual analysis and try to understand the difference b/w training and dev/test sets.
- Make train set more similar.

Learn from Multiple Task.

Transfer Learning :>



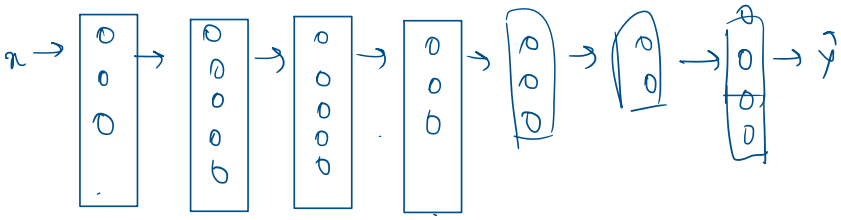
When transfer learning makes sense :>

When transfer learning makes sense :→

Transfer (A) → (B) { They have same input x, A has more data than B. }

Multi-Task Learning.

* Start off simultaneously



Loss $\hat{y}^{(i)} = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{4} \mathcal{L}(\hat{y}_j^{(i)}, y_j^{(i)})$ (Logistic loss).

* Train a set of task that could benefit from having shared low level features.

End-to-End deep learning.

Speech Recognition (ex.)

X → features → phonemes → words → Y

(audio) MFCC (transcript)

end-to-end deep learning

Whether to use End-to-end dep. learning

Pros
- Let the data speak.
- Less hand design components needed.

Cons
- Needs large amount of data.

  X → Y
- Excludes potentially useful hand design components.

Week 1-Foundations of CNN

15 June 2020 10:29

Convolutions Neural Networks

Computer Vision

- Object detection.
- Image classification
- Neural Style transfer.

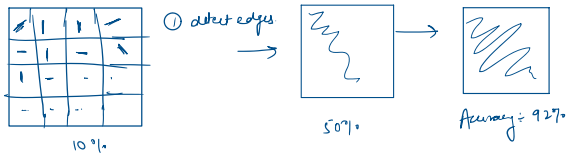
64×64 :- (64×64) × 3
{r g b}

1024 × 1024 (Imgs) :- (1024 × 1024) × 3
≈ 3 million.

(Very high)

Edge Detection.

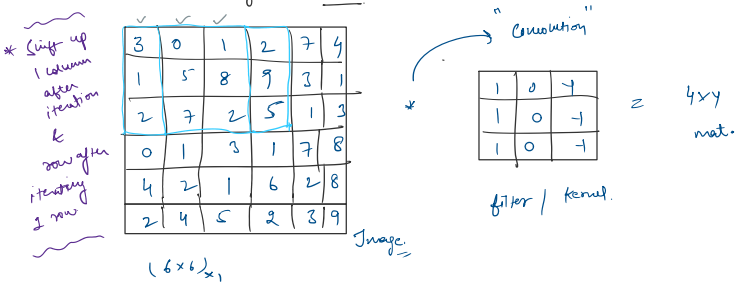
Computer vision problems



① detect Vertical lines. (|)

② detect horizontal lines. (—)

- Vertical edge detection



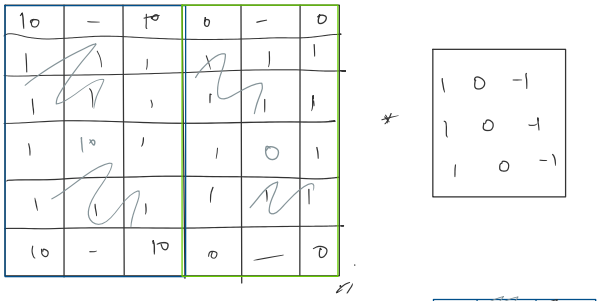
-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

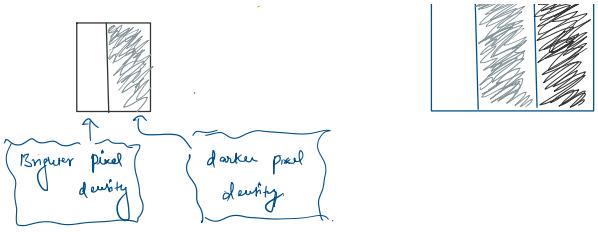
(3*1) + (0*0) + (1*-1) + (1*1) + (5*0) + (8*-1) + (2*1) + (7*0) + (2*-1).
= -5.

(0*1) + (1*0) + (2*-1) + (5*1) + (8*0) + (9*-1) + (7*1) + (2*0) + (5*-1) = -4

* (Image - Vertical Edge detector)

- tf. nn. Conv2d.





(i) pixel value $\propto$ $\frac{\text{bright pixels}}{\text{dark pixels}}$

Horizontal Edge Detection

1	1	1
0	0	0
-1	-1	-1

More filters

1	0	-1
2	0	-2
1	0	-1

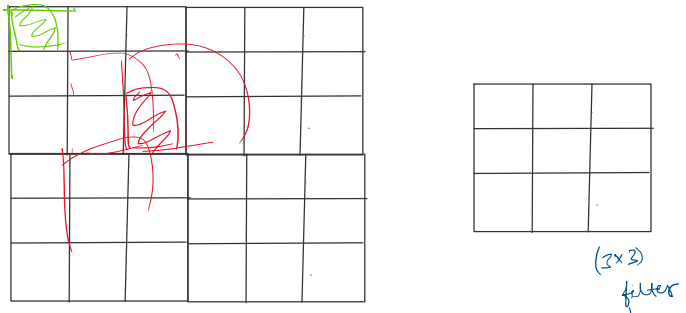
* Sobel filter

3	0	-3
10	0	-10
3	0	-3

* Scharr filter

Vertical Edge detection.

Padding



(6×6) Image

(3×3) filter

(n×n) (f×f)

$(n-f+1 \quad * \quad n-f+1)$

(6,6) $\xrightarrow[\text{shrink}]{}$ (4,4) ⇒ (1,1)

f(3,3).

- Image shrinking.
- Top left pixel (0,0) is used only one but centre pixel is overlapped many times.

- Top left pixel [0,0] is used only one but centre pixel is overlapped many times.
- We are loosing information from corner pixels.

* padding

$(6\times 6) \rightleftharpoons (8\times 8)$

$(8-3+1) = 6\times 6$ image (process image)

- padding of 0s

p: padding = 1 :- $(n-f+1)$

$n(+2p)-f+1$

Valid and Same Convolutions

"valid" :- No padding.

$(n*n) \rightarrow (f*f) \Rightarrow (n-f+1) * (n-f+1)$

"same" :- Pad so that input size is same as output size.

$(n+2p-f+1)$

$n+2p-f+1 = n$

$p = (f-1)/2$

* f is usually odd.

$f \in 2n+1$

- **Strided Convolution**

stride $\rightleftharpoons$ step

stride = 2 (say)

step by (stride)

padding: p
stride: s
image: n.

$\left[\left(\frac{n+2p-f}{s}\right)+1\right] \times \left[\frac{\quad}{\quad}\right]$

↳ if this is not integer then floor $(\mathbb{Z})$

Cross-Correlation v/s Convolution

① Sometimes we flip the filter (vertically and horizontally) and this process is called "Convolution" and process we perform is called Cross-correlation.

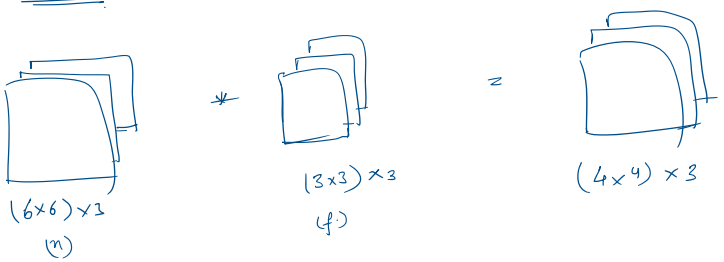
$$f \Rightarrow \begin{bmatrix} 3 & 4 & 5 \\ 1 & 0 & 2 \\ -1 & 9 & 7 \end{bmatrix}$$

$$f* \Rightarrow \begin{bmatrix} 7 & 9 & -1 \\ 2 & 0 & 1 \\ 5 & 4 & 3 \end{bmatrix}$$

Convolution Over Volumes.

Convolution Over Volumes.

① On RGB



height × width × channels

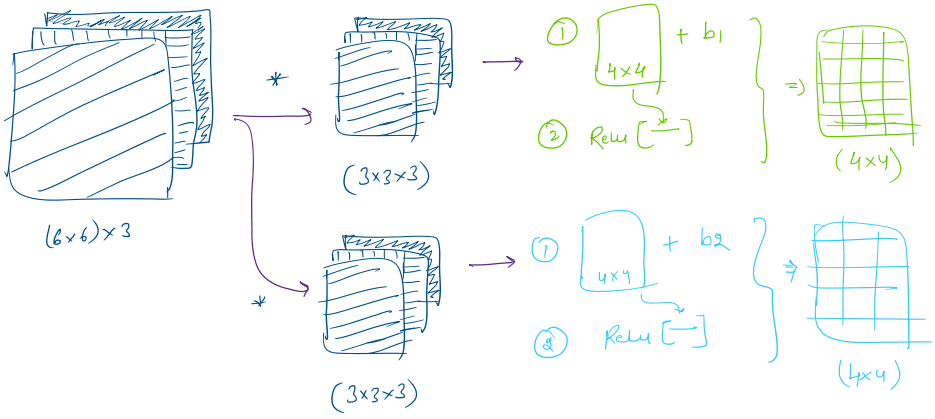
Assert * Channels of Image = Channels on filter.

$(n \times n \times n_c) * (f \times f \times n_c) \Rightarrow (n-f+1) \times (n-f+1) \times n_c'$

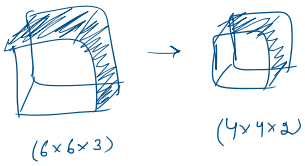
$(6 \times 6) \times 3 \quad * \quad (3 \times 3) \times 3 \qquad 4 \times 4 \times 2$

② Assuming stride = 1, padding = None.

One Layer of CNN.



$\Rightarrow$ $(4 \times 4) \times 2$



1 Layer of ConvNet.

- If layer l is a Convolutional Layer :-

$f^{[l]}$ = filter size.

$p^{[l]}$ = padding

$s^{[l]}$ = Stride

$n_c^{[l]}$ = No. of filters.

Input : $(n_H^{(l-1)} \times n_W^{(l-1)} \times \boxed{n_c^{(l-1)}})$

O/P : $(n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]})$

$n_{(H/W)}^{[l]} = \left[\frac{n^{(l-1)} + 2p^{(l)} - f^{(l)}}{s} + 1 \right]$

$f^{[l]}$ = filter size.

$p^{[l]}$ = padding

$s^{[l]}$ = Stride

$n_c^{[l]}$ = No. of filters.

Size of filter :

$f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Input : $\left(n_H^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]} \right)$

O/P : $\left(n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]} \right)$

$n^{[l]}_{(H/w)} = \left[\frac{n^{[l-1]} + 2p^{[l]} - f^{[l]}}{s} + 1 \right]$

Activations :

$a^{[l]} \rightarrow n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

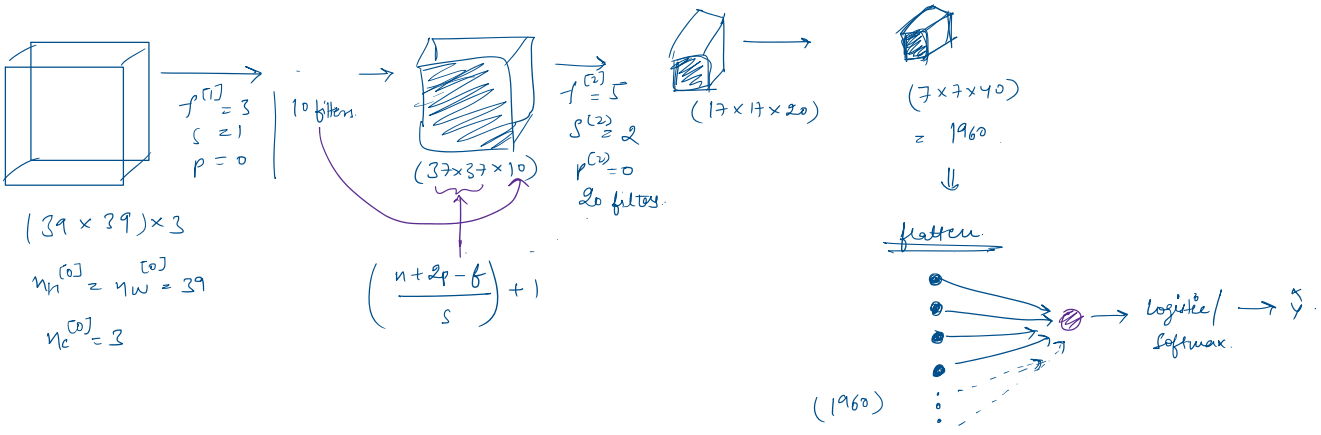
$A^{[l]} \rightarrow m \times a^{[l]}$

Weights :

$\underbrace{\left(f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \right)}_{1 \text{ filter}} \times \underbrace{n_c^{[l]}}_{(\#\text{filters})}$

bias : $n_c^{[l]}$ $(1 \times 1 \times 1 \times n_c^{[l]})$

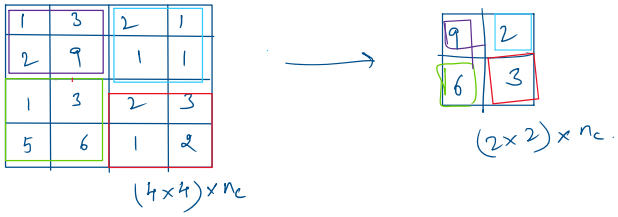
Simple Convolution N/w Example.



Types of N/w in CNN :

- Convolution. (CONV)
- Pooling Layer (POOL)
- Fully Connected (FC)

Pooling Layers. : Max pooling



Hyperparameters ⇒

* filter, $f = 2$
* stride, $s = 2$

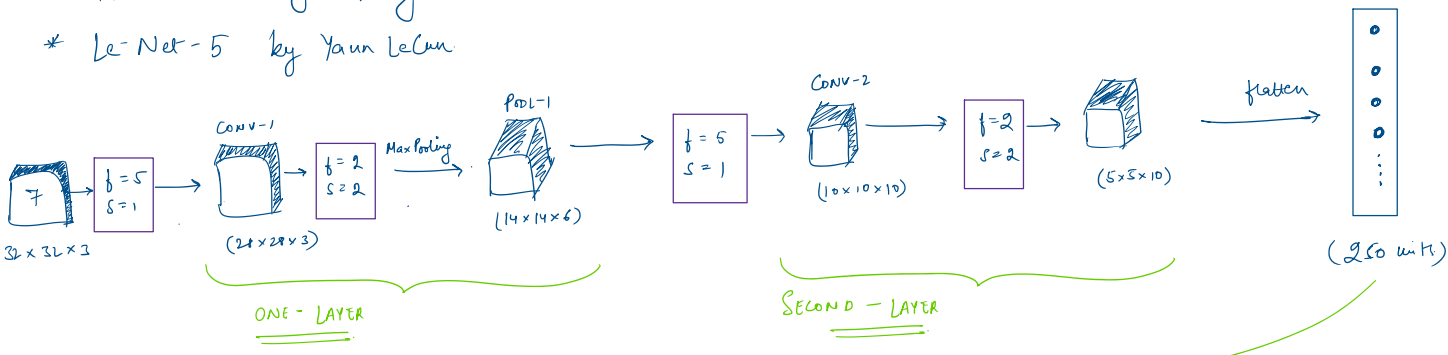
* Take max over 2×2 regions.
* Takes no parameters.

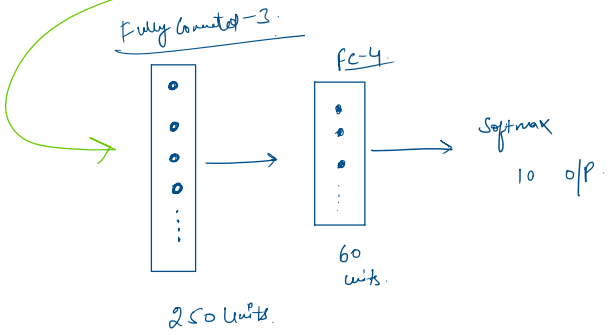
$$\left(\frac{n + 2p - f}{s}\right) + 1$$

- Less used : Average pooling.

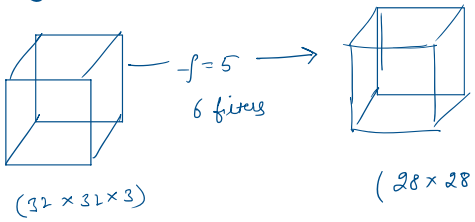
CNN Example

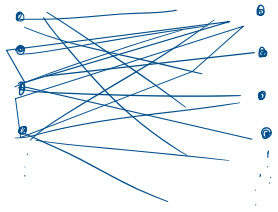
- Handwritten digit recognizer.
- Le-Net-5 by Yann LeCun





Why Convolutions. ?





3072 × 4704

≈ 14 million.

#1 Parameter Sharing :-

A feature detector that's useful in one part of the image is probably useful in another part of image

#2 Sparsity of Connections :-

In each layer, each O/P value depends only on a small number of units.

1 O/P unit is connected with 6/36 units.

(n) (n×n)

Week 2- deep convolution models: Case Study?

18 June 2020 12:23

Classic N/w's

- Le- Net 5
- AlexNet
- VGG-16.

→ ResNet (Residual N/w).

152 layer nn.

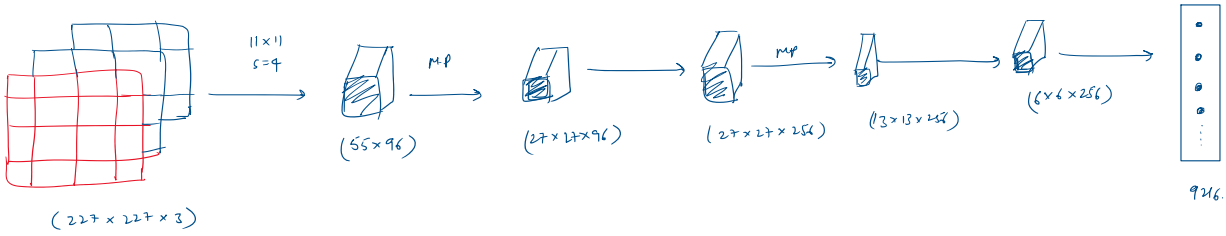
→ Inception.

- **Le Net - 5**

60k params.

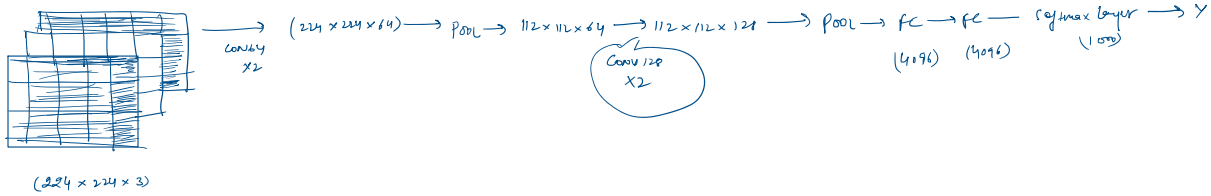
conv → pool → conv → pool → fc1 → fc2 → ŷ.

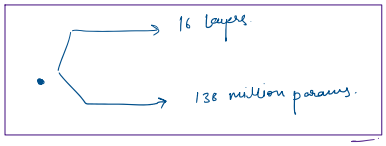
AlexNet



VGG - 16

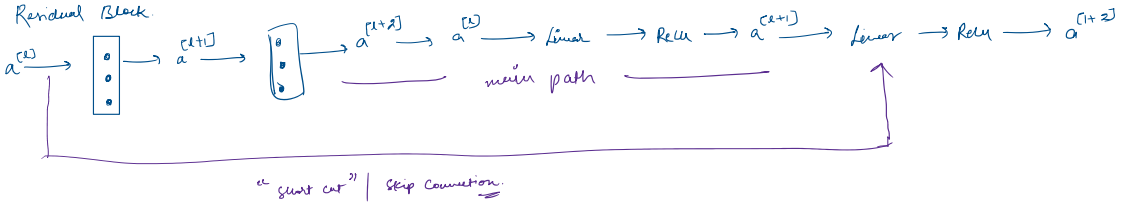
CONV = 3x3 filter, s=1, same.
MAX POOL = 2x2, s=2.

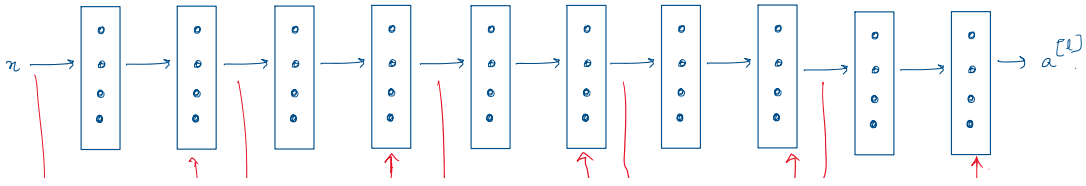


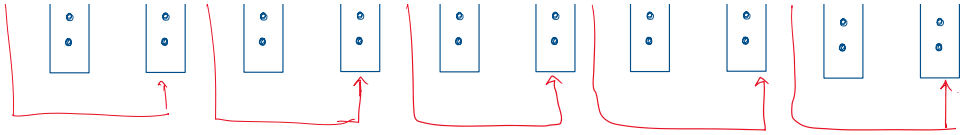


ResNets - Residual N/w's

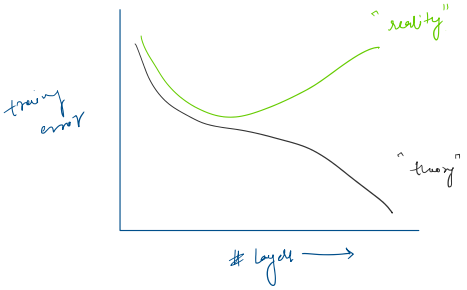
Residual Block.



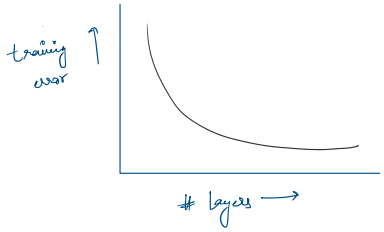




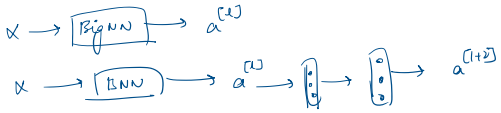
Plain.



ResNet



Why ResNets Work ?



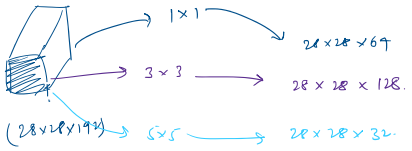
$\therefore$ (ResNet)

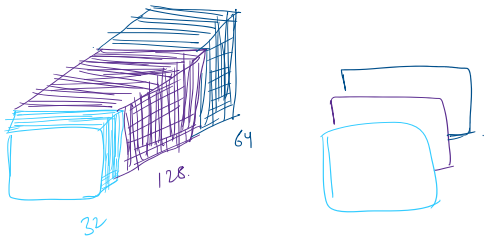
$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$

$z = wa + b$

Inception N/W.

What to Choose?
Pooling layer / Conv. layer ?
No. of filters ?





(X) Computational Cost

(28 × 28 × 192) → CONV 5×5, same, 32 → (28 × 28 × 32)

$(28 \times 28 \times 32) \times (5 \times 5 \times 192)$

$\approx$ 120 million.

Reduce Cost by 1×1 Convolutions !

- $(28 \times 28 \times 192)$ —CONV. 1×1, 16, 1×1×192→ $(28 \times 28 \times 16)$ —CONV, 5×5, 32, 5×5×16→ $(28 \times 28 \times 32)$

Also called "Bottleneck Layer".

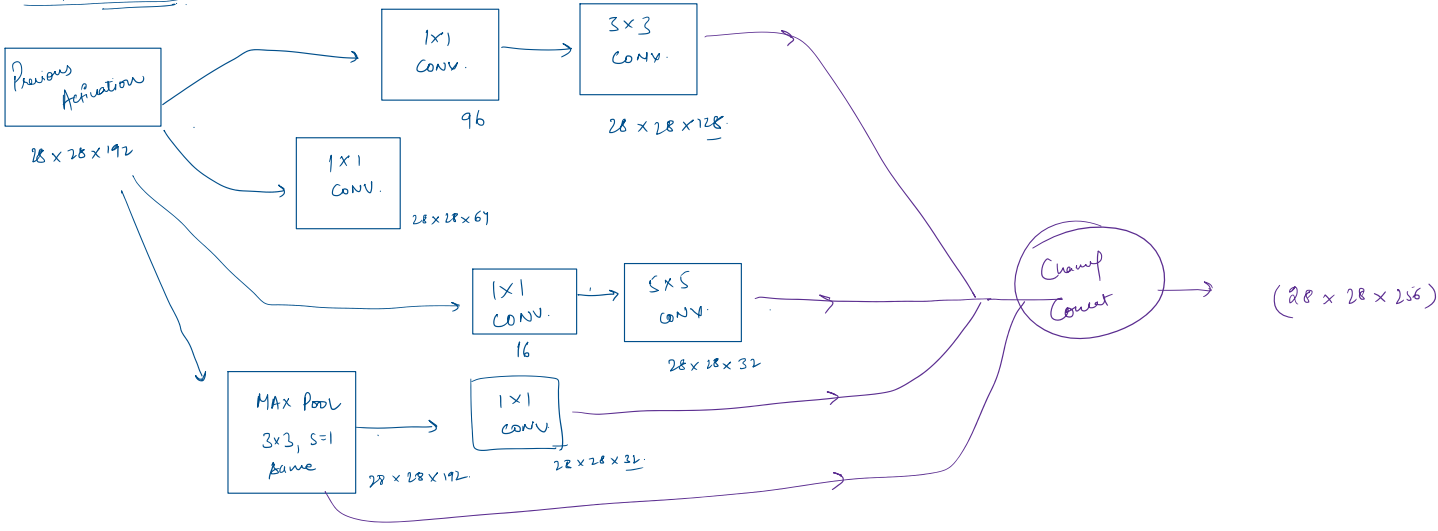
Convolutional Cost :→ $(28 \times 28 \times 16) \times 192 \approx 2.4$ million

+

$(28 \times 28 \times 32) \times (5 \times 5 \times 16) \approx 10$ million

$\approx$ 12.4 million

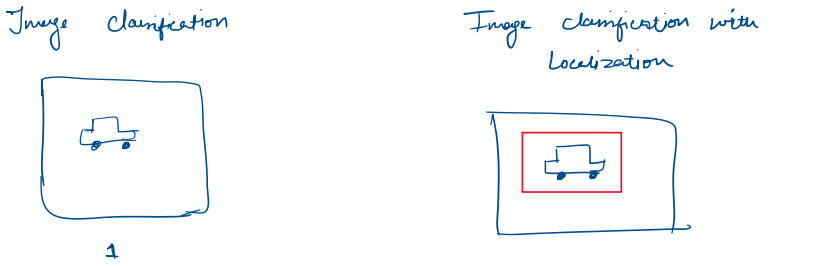
Inception N/W

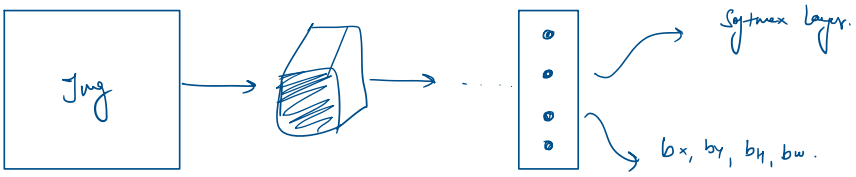


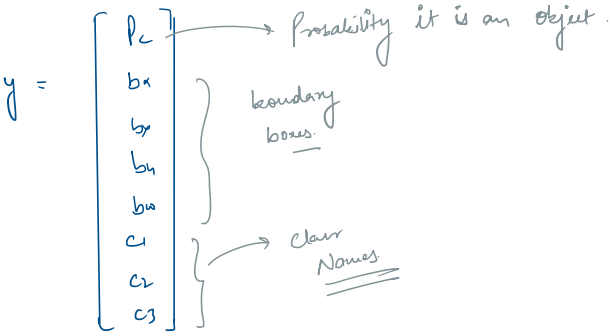
Week 3 - Detection Algorithms

19 June 2020 22:36

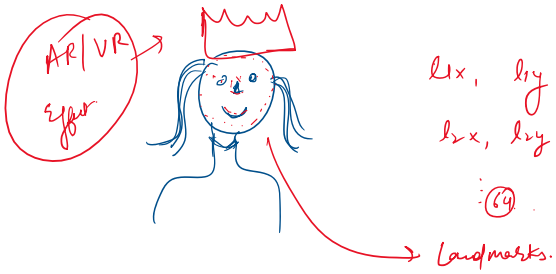
Object Localization





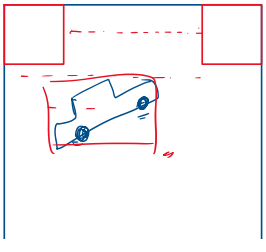


Landmark Detection



Object Detection

Sliding window detection



Traverse with whole window.

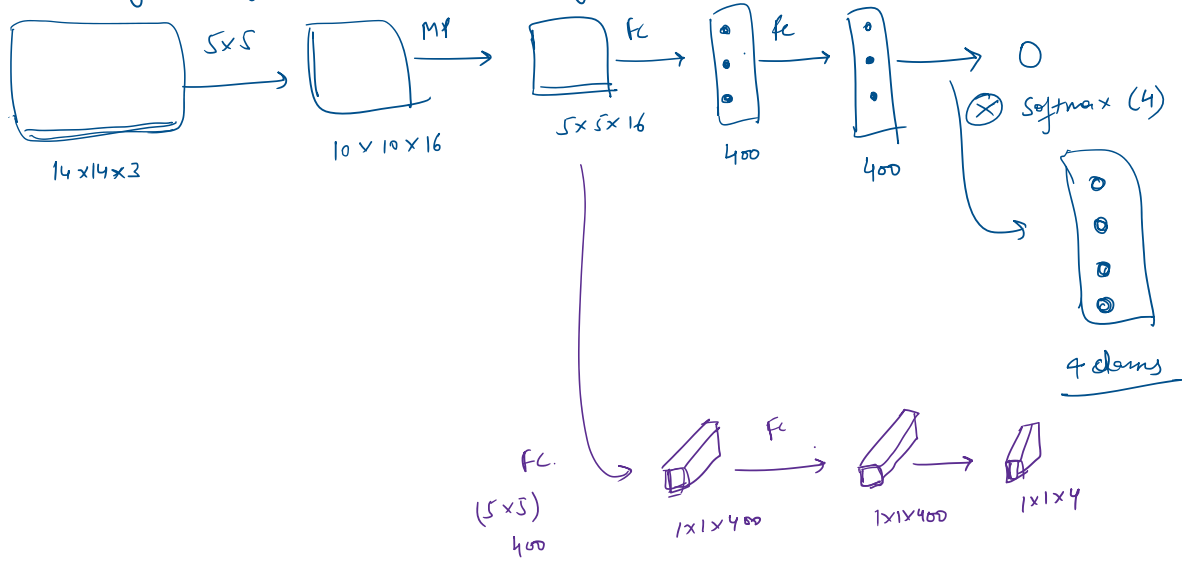
increase the window size.

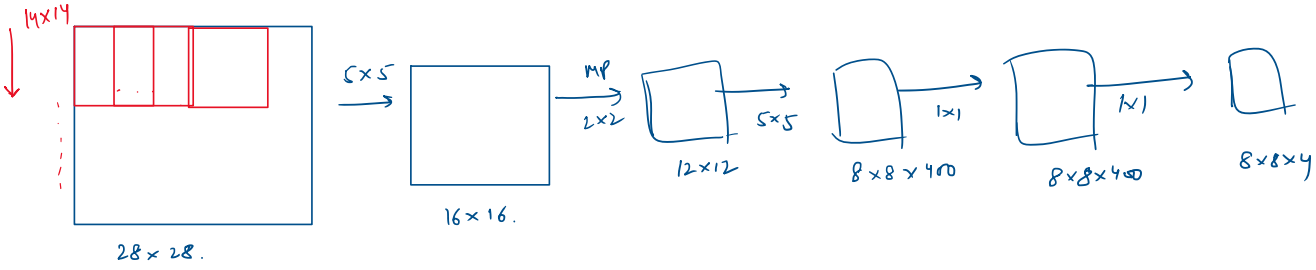
- Very Expensive Computational Cost

- Very Expensive Computational Cost

Convolution implementation of Sliding Window.

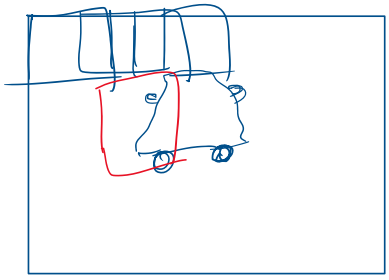
- Turning fc layers into Convolutional layers.





Bounding box prediction.

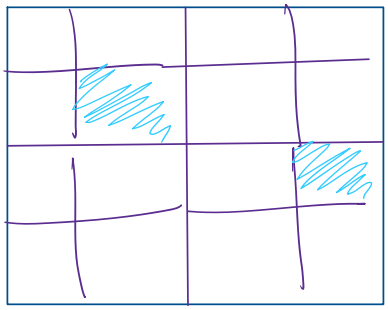
- Output accurate bounding boxes.



* None of the bounding box is Accurate!

YOLO Algorithm.

→ "You only look once"



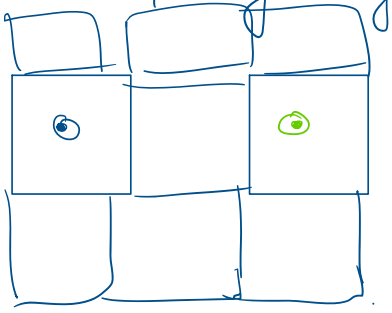
* divide into grids.
* Object to be detected.

Labels for training

for each grid cell -

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_H \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

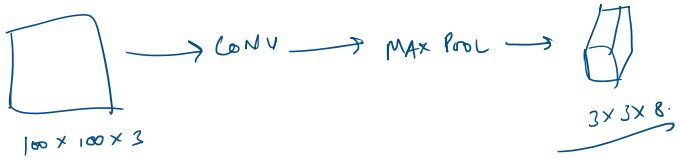
① Mid point of each grid cell is assigned if object is detected.



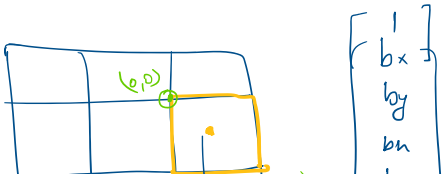
Total Volume of O/P :->

3 × 3 × 8

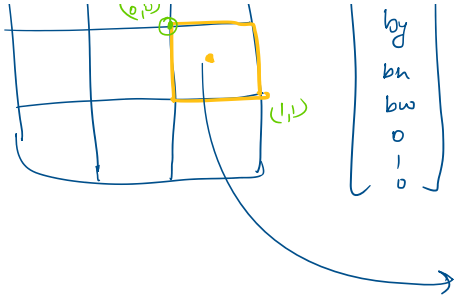
8 dimensional o/p vector.



Specifying the bounding box.



$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_H \end{bmatrix}$$



b_y
b_h
b_w
0
1
0

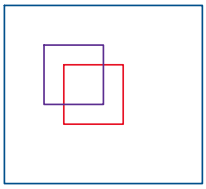
$b_x = 0.7$
$b_y = 0.3$
$b_w = 0.9$
$b_h = 0.5$

$0 < x < 1$

could be > 1

INTERSECTION OVER UNION.

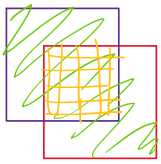
Evaluating Object Localization



□ : ground truth bounding box.

□ : Algorithm Output.

IOU



∪ Union

∩ Intersection

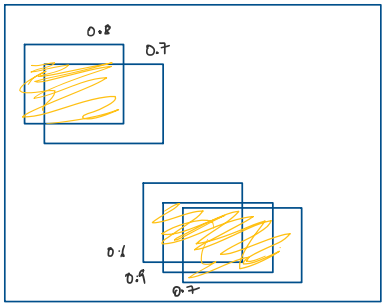
$$\frac{\text{Size of intersection } \cap}{\text{Size of union } \cup}$$

"Correct if IOU ≥ 0.6"

(*) IOU is the measure of Overlap.

NON-MAX SUPPRESSION.

(X) Algorithm can detect same object multiple times, NMS makes sure that object is detected only once.

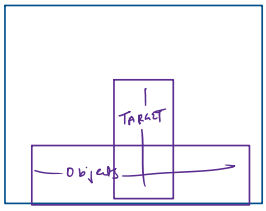


$p(c)$ = Probability of Object.

- Discard all boxes with $p(c) \leq 0.6$ with the remaining boxes :→
- Pick the boxes with maximum $p(c)$.
- discard any remaining box with IOU ≥ 0.5 with the box output in the previous step.

ANCHOR BOXES.

(X) Only one object in the gridcell can be identified



predefine two (n) Anchor Boxes

- Anchor Box 1 
- Anchor Box 2 

$y =$

P_c
b_x
b_y
b_h
b_w
c_1
c_2
c_3
P_c
b_x
b_y
b_h
b_w
c_1
c_2
c_3

Anchor Box 1.

Anchor Box 2

$y = 3 \times 3 \times 16$

$(2 \times n)$

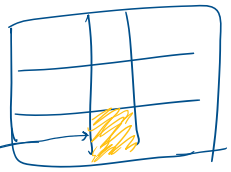
$(n = 2)$

YOLO ALGORITHM : Putting it together.

Training 

(1) pedestrian.

Training



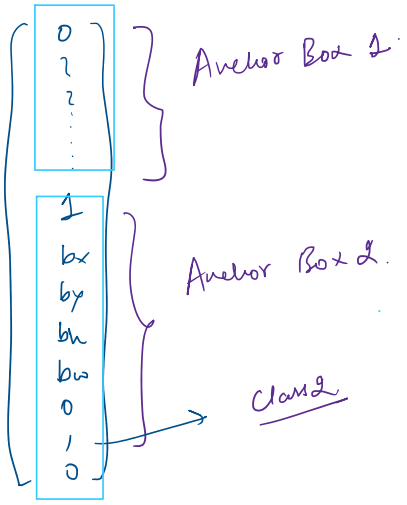
1. pedestrian.
2. car
3. motorcycle.

$y = 3 \times 3 \times 2 \times 8 \longrightarrow$ # classes + 5

↳ # Anchors

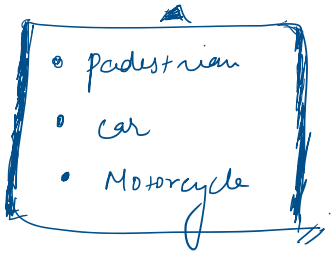
Anchor #1

Anchor #2



Outputting the non-max Supprend O/P :→

- For each grid cell, get two predicted bounding boxes.
- Get rid of low probability predictions.
- For each class, use non-max Suppression to generate final predictions.



Week 4 - Special applications: Face recognition & Neural style transfer

21 June 2020 19:31

FACE RECOGNITION.

- Verification
  - → Input image.
  - → O/p whether the image is of claimed person (0/1).
- Recognition.
  - → Has database of K persons.
  - → get an input image.
  - → O/p Id of that person if in db.

ONE SHOT LEARNING.

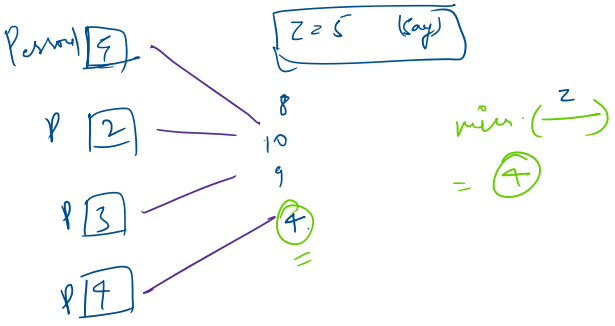
"Learn by only single input image".

* Learn similarity fx.

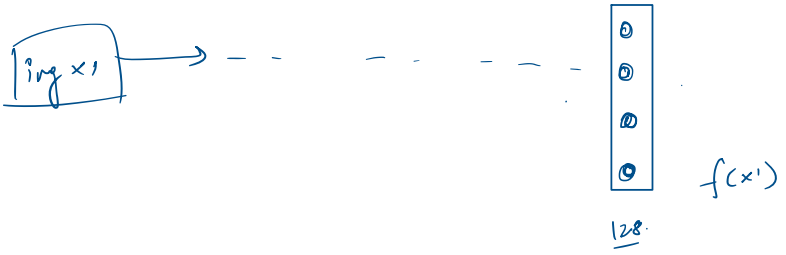
d(img1, img2) = degree of difference b/w images img1 and img2.

if { d ≤ τ "Same" ; d > τ "different" }

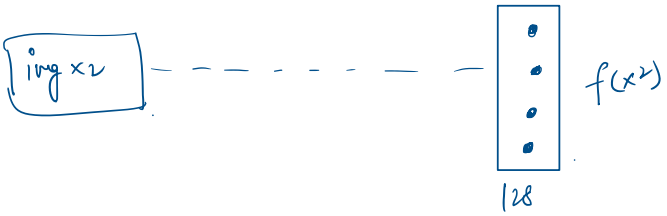
- τ = Tau (some hyperparameter)



SIAMESE NETWORK.



encoding of $img(x_1) = f(x_1)$.

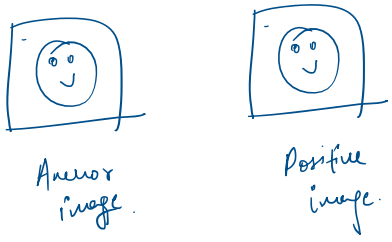


$$d(x_1, x_2) = \| f(x_1) - f(x_2) \|_2^2$$

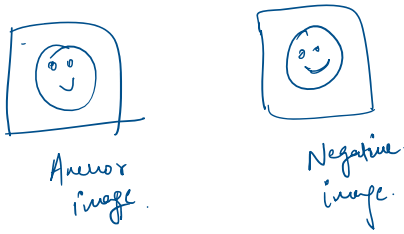
Comparing two images and calculating d is sometimes called "Siamese Network".

"Deep Face".

TRIPLET LOSS Fn.



$z \ll$



$z \gg$

Triplet : { Anchor, A ; +ve, P ; -ve, N.

- $\underbrace{\| f(A) - f(P) \|^2}_{d(A,P)} \leq \underbrace{\| f(A) - f(N) \|^2}_{d(A,N)}$

$$d(A,P) - d(A,N) \leq 0$$

$$\| f(A) - f(P) \|^2 - \| f(A) - f(N) \|^2 \leq 0 - \alpha$$

(α : hyperparameter)

or

$$\left[\| f(A) - f(P) \|^2 - \| f(A) - f(N) \|^2 \right] + \underbrace{\alpha}_{\text{(Margin.)}} \leq 0$$

$\alpha = 0.2$ (say)

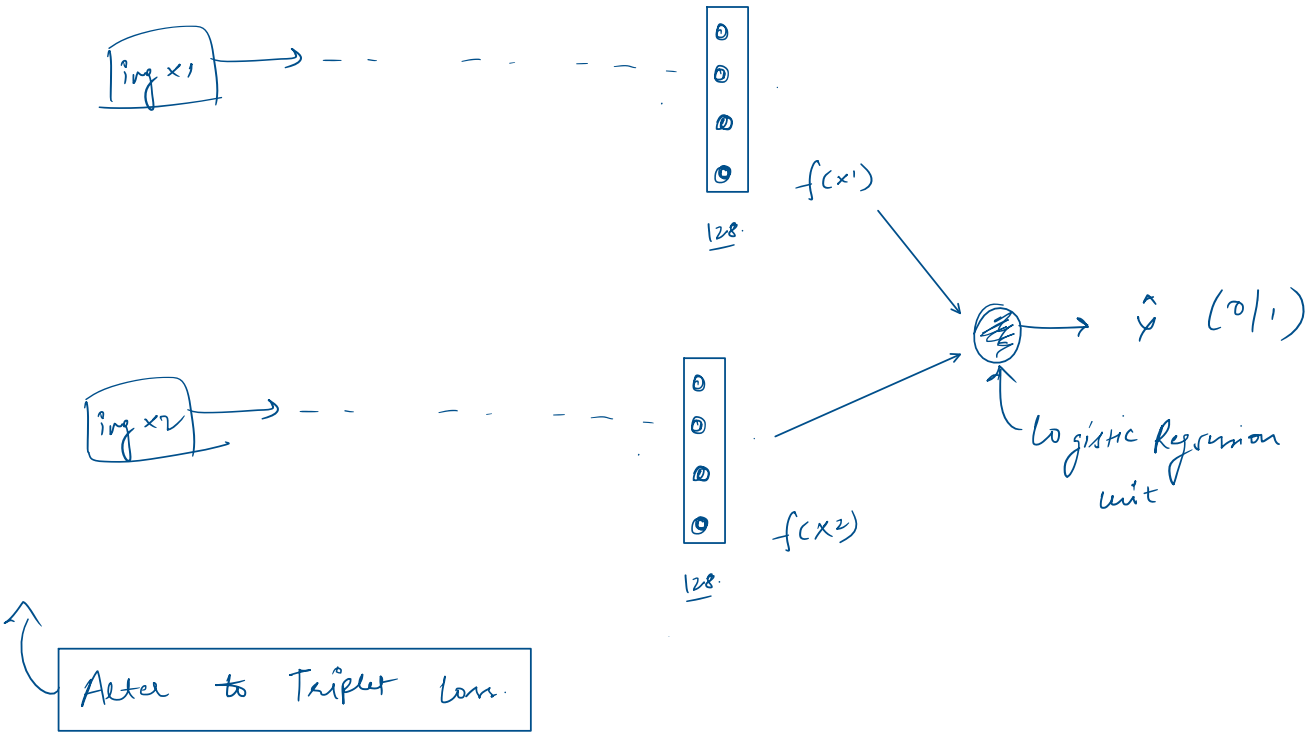
Loss fx.

- given three images A, P, N.

$$\mathcal{L}(A,P,N) = \max \left(\underbrace{\| f(A) - f(P) \|^2 - \| f(A) - f(N) \|^2 + \alpha}_{\leq 0}, 0 \right)$$

$$J = \sum_{i=1}^{m} \mathcal{L}(A_i, P_i, N_i)$$

Face Verification & binary Classification.



Alter to Triplet loss.

$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_k \left| f(x^{(i)})_k - f(x^{(j)})_k \right| + b\right)$$

NEURAL STYLE TRANSFER

Image 1 (Original) + Image 2 (filter Image) ⇒ Neural style transfer.

⊙ deep Convnets learning.

Pick a unit in Layer 1. Find the 9 image patches that maximize the units activation. Repeat for other units.

Cost fx.

Content C | Style S

→ generated image G

$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$

$J_{content}(C, G)$ → similarity.

Initiate G randomly,

G : 100 × 100 × 3

$G := G - \frac{\partial}{\partial G} J(G)$.

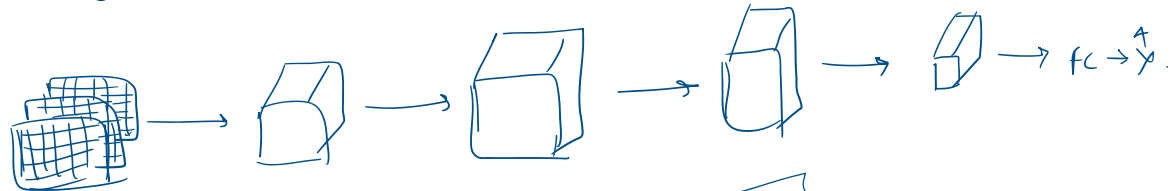
Content Cost fx.

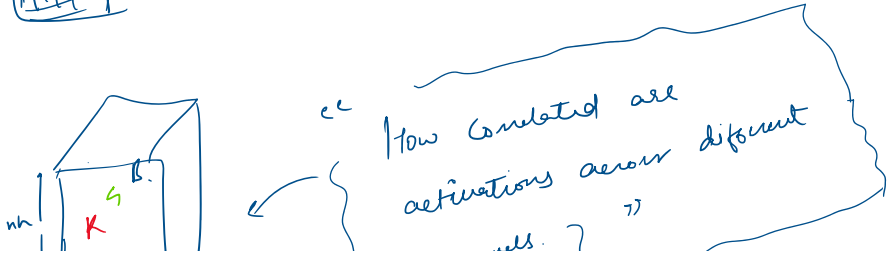
$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$.

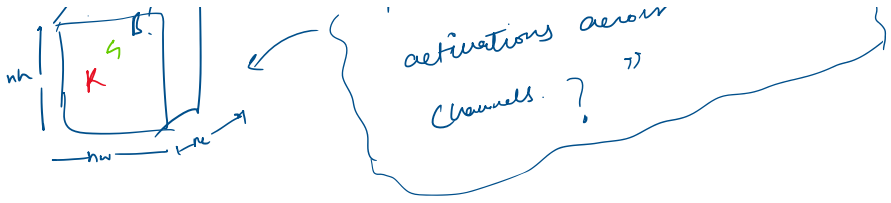
- Say you use layer l.
- Use pre-trained ConvNet (say VGG).
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be activation on layer l on the image.
- If $a^{[l](C)}$ & $a^{[l](G)}$ are similar, both images have similar content.

$$J_{content}(C, G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|^2$$

Style Cost fx.







Style Matrix

Let $a^{[l]}_{ijk}$ = activation at (i, j, k)

$H \quad W \quad C$

$G^{[l]}$ is $n_c^{[l]} \times n_c^{[l]}$.

$$G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} \left(a^{[l](G)}_{ijk} \cdot a^{[l](G)}_{ijk'} \right)$$

(Correlation)

$$G^{[l](S)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} \left(a^{[l](S)}_{ijk} \cdot a^{[l](S)}_{ijk'} \right)$$

G = generated image

S = Style image

"gram matrix".

$$\boxed{J^{[l]}_{style}(S, G) = \| G^{[l](S)} - G^{[l](G)} \|^2_F}$$

$$= \frac{1}{(2 n_H^{[l]} n_W^{[l]} n_c^{[l]})^2} \sum_k \sum_{k'} \left(G^{(l)(S)}_{kk'} - G^{(l)(G)}_{kk'} \right)$$

$$J(G) = \alpha \left[J_{content}(C, G) \right] + \beta \left[J_{style}(S, G) \right]$$

1d and 3d generalization

2d image (14 × 14) * 2d filter (5 × 5) = (10 × 10)

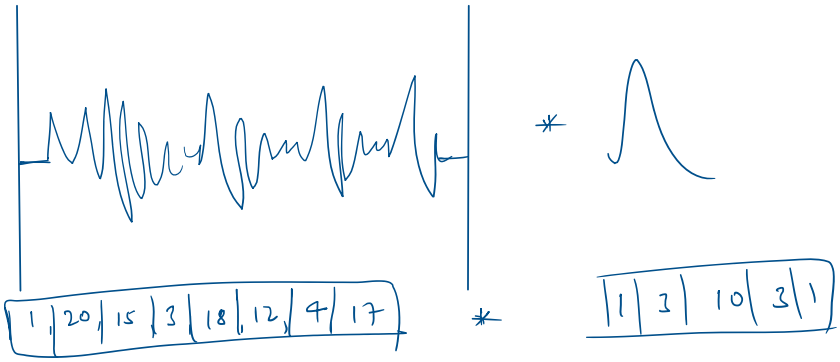
2d image * 2d filter = (10 × 10)

(14 × 14) (5 × 5)

Same can be applied to 1d image.

ex. ECG

electro cardio graphy.

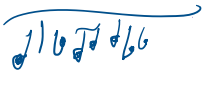


Why Sequence Models ?

- Examples of Sequence data :-

  X → Y

  ① Speech recognition :-  → "The quick brown fox jumps over the lazy dog".

  ② Music generation :- ϕ → 

  ③ Sentiment classification :- "bad movie" → 1 ★

  ④ DNA Sequence Analysis → AGCCCTGTACCG

  ⑤ Machine translation → "Hermosa" → "Beautiful"

  ⑥ Video activity recognition →  → "Running"

  ⑦ Name entity recognition → "Yesterday Harry potter met Hermoine granger."

NOTATIONS

ex.

X : "Harry Potter & Hermoine granger invented a new Spell".

Name entity recognition

Y : 1 1 0 1 1 0 0 0 0

- input is sequence of 9 words, 9 sets of features to represent these 9 words.

$x^{<t>}$, $x^{<1>}$, $x^{<2>}$ ---- ---- $x^{<9>}$ $T_x = 9$

"temporal Sequence"

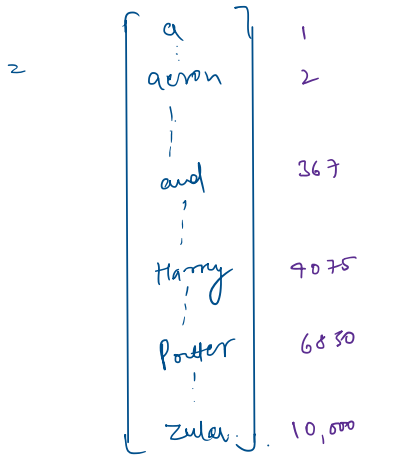
$y^{<t>}$, $y^{<1>}$ $y^{<2>}$ ---- $y^{<9>}$ $T_y = 9$

$x^{(i)<t>}$, $T_x^{(i)}$

$x^{(i)<T>}$, $Tx^{(i)}$

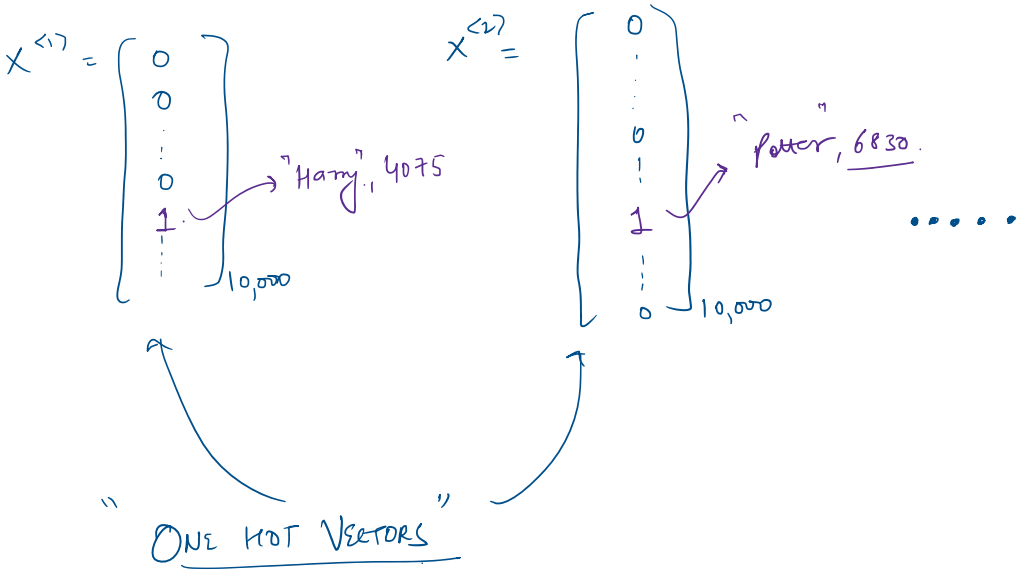
$y^{(i)<T>}$, $Ty^{(i)}$

Vocabulary / dictionary



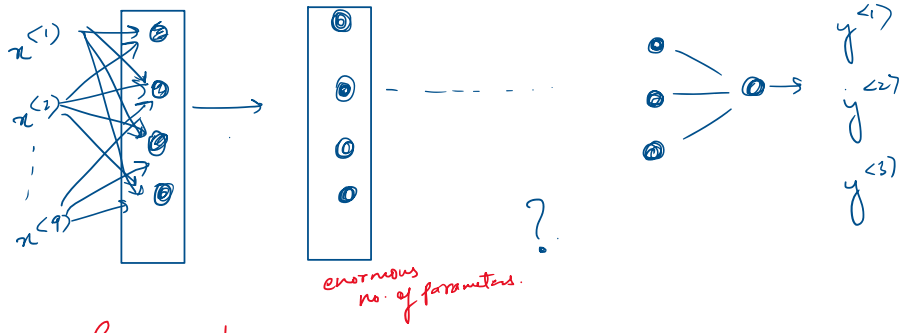
(Assuming).

One Hot representation



RECURRENT NEURAL N/W MODEL.

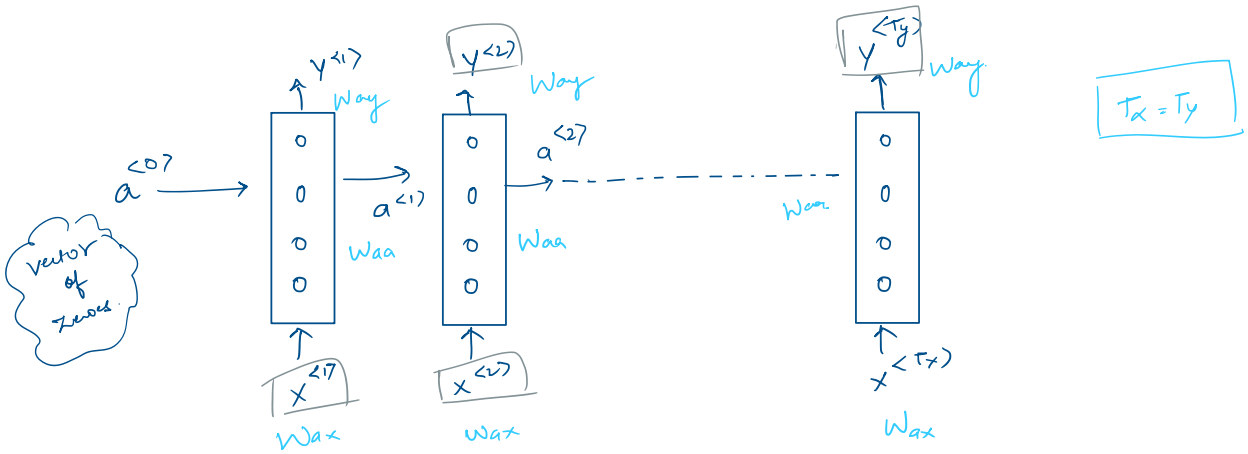
Why not a Standard Neural N/W ?



Problems!

1. I/p and O/p can be different lengths in diff. example.
2. doesn't share features learned across different position of text.

Recurrent Neural N/W

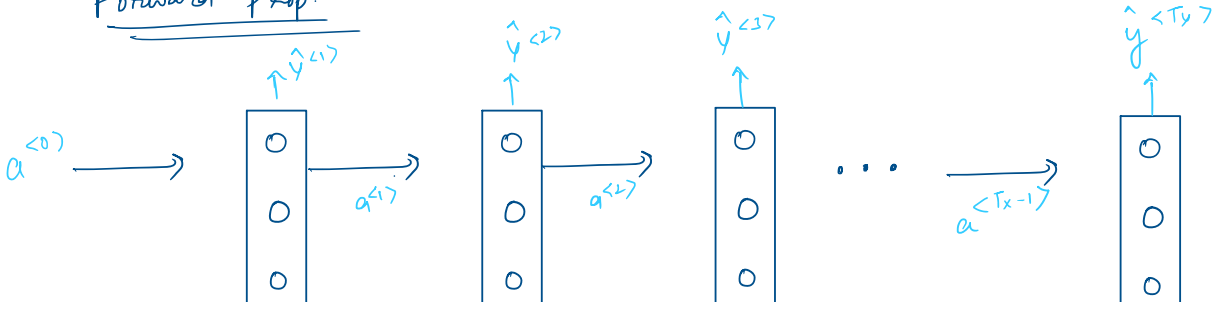


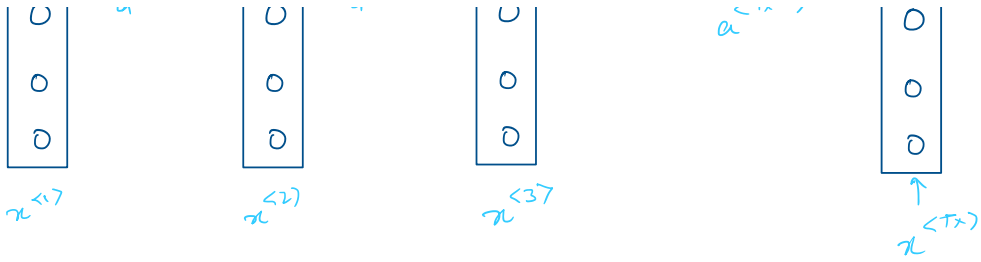
> While making prediction $y^{<2>}$, it also uses $x^{<2>}$ and $x^{<1>}$ or (earlier in the sequence.)

ex. He said "Teddy Roosevelt was a great president." ✓

He said "Teddy Bears are on Sale" ✗

Forward Prop.





$a^{<0>} = \vec{0}$

$$a^{<1>} = g_1(W_{aa} \cdot a^{<0>} + W_{ax} \cdot x^{<1>} + b_a) \quad \rightarrow \tanh(\text{tw}) \,/\, \text{ReLU}$$

$$\hat{y}^{<1>} = g_2(W_{ya} \cdot a^{<1>} + b_y) \quad \rightarrow \text{Sigmoid}$$

$$a^{<t>} = g(W_{aa} \cdot a^{<t-1>} + W_{ax} \cdot x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya} \cdot a^{<t>} + b_y)$$

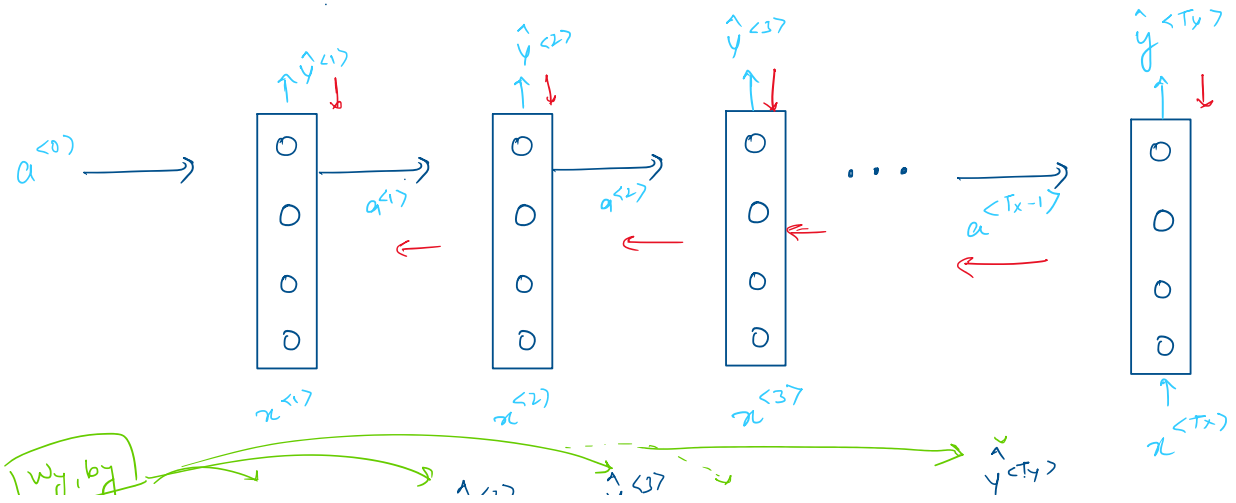
Simplified RNN Notation

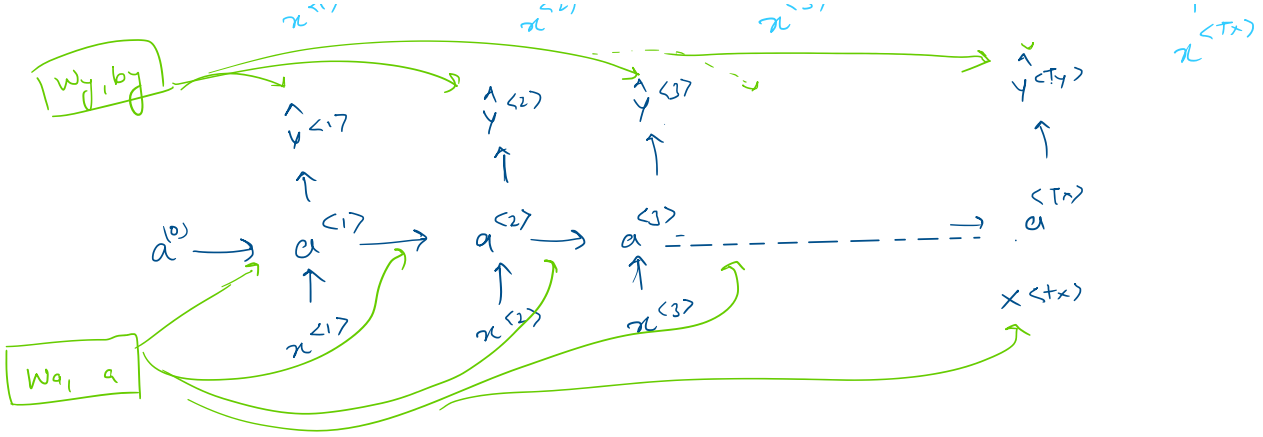
$$a^{<t>} = g(W_{aa} \cdot a^{<t-1>} + W_{ax} \cdot x^{<t>} + b_a) \iff g(W_a(a^{<t-1>} + x^{<t>}) + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya} \cdot a^{<t>} + b_y)$$

$$[W_{aa} \,|\, W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa}\, a^{<t-1>} + W_{ax}\, x^{<t>}$$

Backprop. Through Time



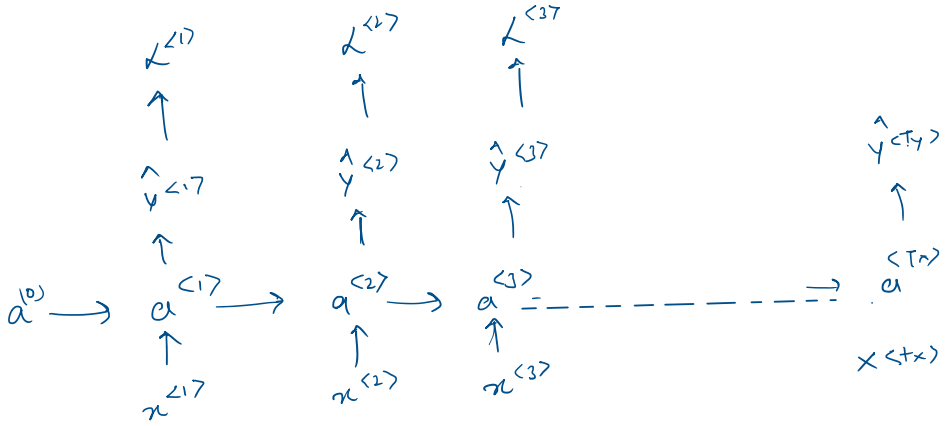


Loss fx.

$$\mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>}\log\hat{y}^{<t>} - (1-y^{<t>})\log(1-\hat{y}^{<t>})$$

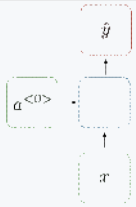
"Cross entropy loss"

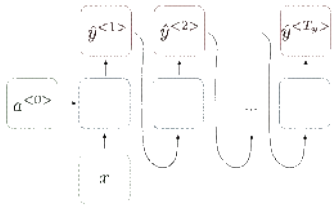
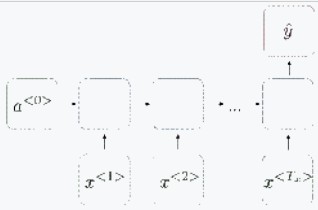
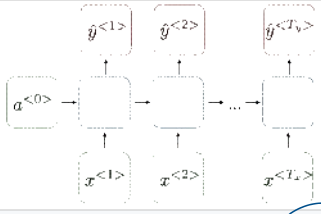
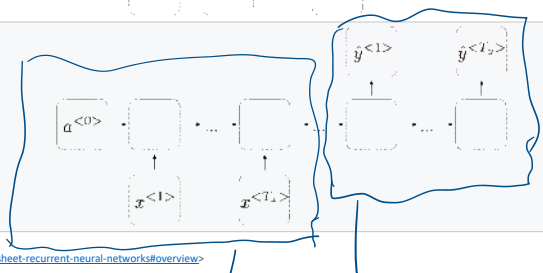
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{Tx} \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$



Different Types Of RNN.

when $Tx \neq Ty$.

Type of RNN	Illustration	Example
One-to-one Tx = Ty = 1		Traditional neural network

One-to-many Tx = 1, Ty > 0		Music generation
Many-to-one Tx >1 , Ty=1		Sentiment classification
Many-to-many Tx = Ty		Name entity recognition
Many-to-many Tx != Ty		Machine translation

From <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#overview>

Stanford University CS230

"Encoder"

"Decoder"

Language Model And Sequence Generation.

Speech Recognition.

"The apple and pear salad."

pear / pair. ?

- P(The apple and pear salad) = 5.7×10^{-10} ✓

P(The apple and pair salad) = 3.2×10^{-13}.

P(Sentence) = ?

$y^{<1>}, y^{<2>}, y^{<3>}, \ldots\ldots\ldots y^{<T_y>}$

Language Building with RNN.

Training set :- Large Corpus of english text.

collection

Example Sentence :-

Cats average 15 hours of sleep a day.

① Tokenize (Create Vocabulary / dictionary).

$y^{<1>}, y^{<2>}, y^{<3>} \ldots\ldots\ldots y^{<T_y>}$ <EOS>

append an extra token at the end of sequence <EOS>

end of sentence.

"The Egyptian Mau is a breed of cat."

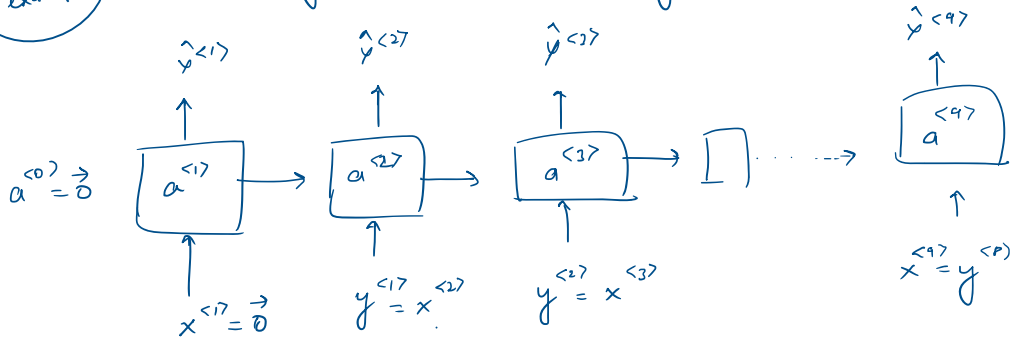
if the word is not in vocabulary replace with '<UNK>' token.

unknown word.

RNN Model

above example

• Cats average 15 hours of sleep a day. <EOS>



$$\mathcal{L}(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$$

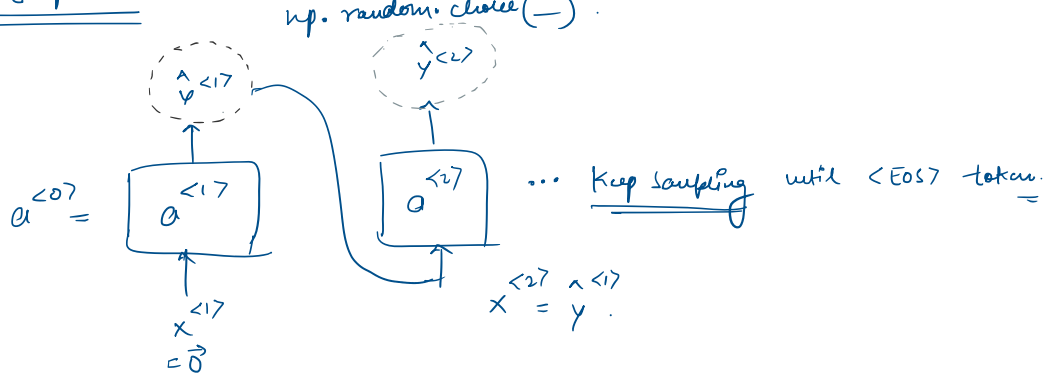
$$\mathcal{L} = \sum_t \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

SAMPLING NOVEL SEQUENCES.

$P(\hat{y}^{<1>}, \hat{y}^{<2>}, \hat{y}^{<3>} \dots \hat{y}^{<T_x>})$

Sample

np.random.choice(—)



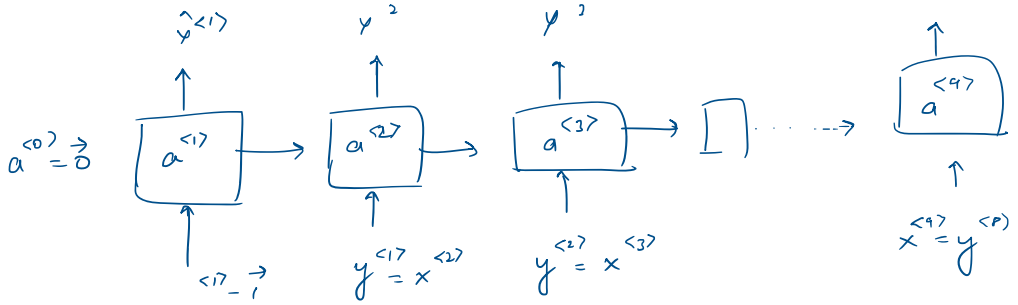
... Keep sampling until <EOS> token.

Character level words —

Vocabulary = [a, b, c ... z, A-Z, ., ,, ;, :, "- - -]

Very expensive.

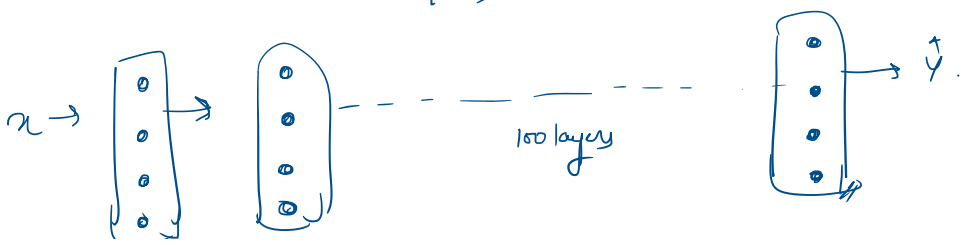
VANISHING GRADIENTS WITH RNNS



Sentence : "The cat which already ate - - - - - - - - , was full."

Cat is singular so
Cat : was
Cats : were

ex. in deep neural nfw :→ (100 layers)



forward prop. →

← Backprop.

(*) if the layer is very deep, we will have hard time calculating gradient, affecting weights of earlier layers.

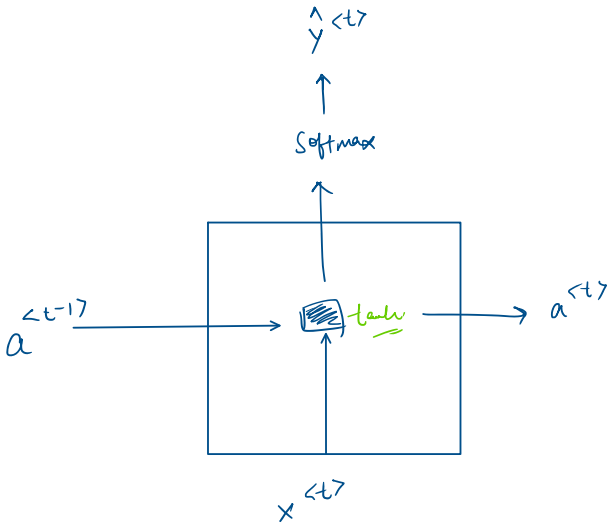
Vanishing / Exploding gradients → by tuning week 1.

→ gradient clipping.

GATED RECURRENT UNIT (GRU)

tanh

$$a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$$



"The cat which already ate ... , was full."

Cat : was
Cats : were

C = memory cell. } Provide memory to remember that the "cat" was singular / plural"

$c^{<t>}$

c = memory cell.

$c^{\langle t \rangle}$, at time t

} Provide memory to remember that the "cat was singular / plural for further consideration.

$c^{\langle t \rangle} \xrightarrow[o/p]{} a^{\langle t \rangle}$

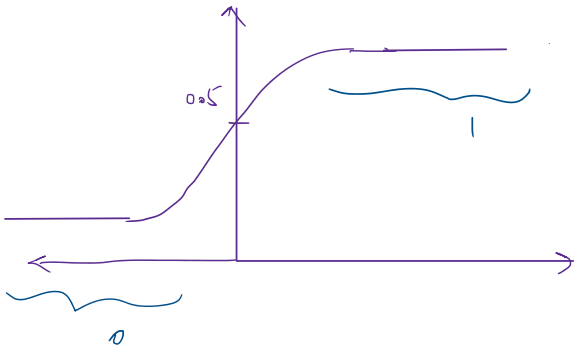
$\tilde{c}^{\langle t \rangle} = \tanh\left(W_c \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right)$.

Candidate for replacing $c^{\langle t \rangle}$

update gate

$\Gamma_u = (0,1) = \sigma\left(W_u \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right)$.

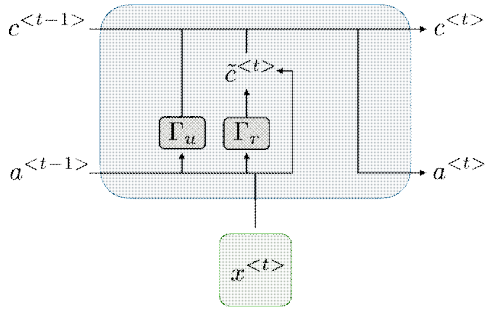
(gamma)



{
singular : 1,
plural : 0.
}

$$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) * c^{\langle t-1 \rangle}$$

FULL GRU



h $c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) * c^{\langle t-1 \rangle}$

"relevance"

$\tilde{h}$ $\tilde{c}^{\langle t \rangle} = \tanh\left(W_c\left[\Gamma_r * c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right)$

r $\Gamma_r = \sigma\left(W_r\left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_r\right)$

u $\Gamma_u = (0,1) = \sigma\left(W_u\left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_u\right)$

$$a^{\langle t \rangle} = c^{\langle t \rangle}$$

LONG-SHORT TERM MEMORY. (LSTM)

$$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) * c^{\langle t-1 \rangle}$$

$$\tilde{c}^{\langle t \rangle} = \tanh\left(W_c\left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right)$$

$$\Gamma_r = \sigma\left(W_r\left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_r\right)$$

$$\Gamma_u = (0,1) = \sigma\left(W_u\left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_u\right)$$

$$\Gamma_f = \sigma\left(W_f\left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_f\right)$$

"forget gate"

$$\Gamma_o = \sigma\left(W_o\left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_o\right)$$

"output"

$$c^{\langle t \rangle} = \left[\Gamma_u * \tilde{c}^{\langle t \rangle}\right] + \left[\Gamma_f * c^{\langle t-1 \rangle}\right]$$

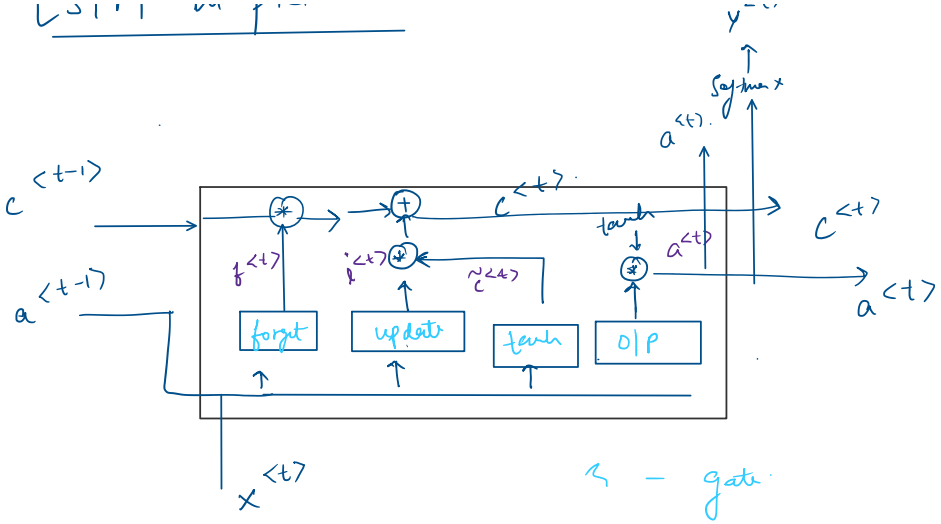
$$a^{\langle t \rangle} = \Gamma_o * \tanh\left(c^{\langle t \rangle}\right)$$

LSTM in picture

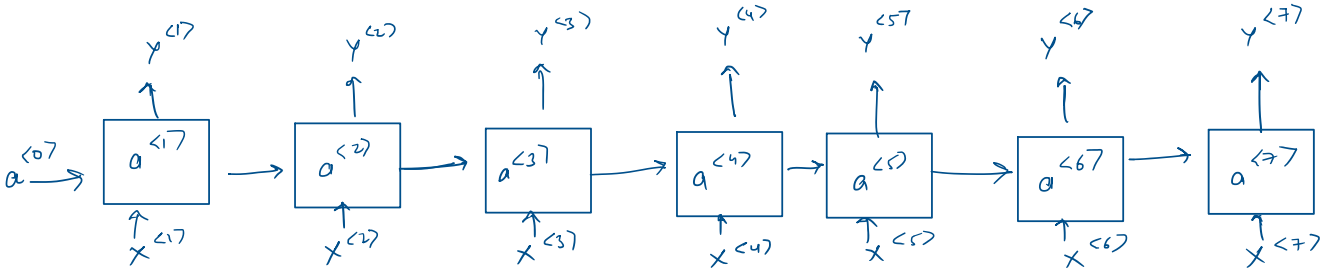
$y^{\langle t \rangle}$

↑

Softmax



BIDIRECTIONAL RNN. (BRNN)

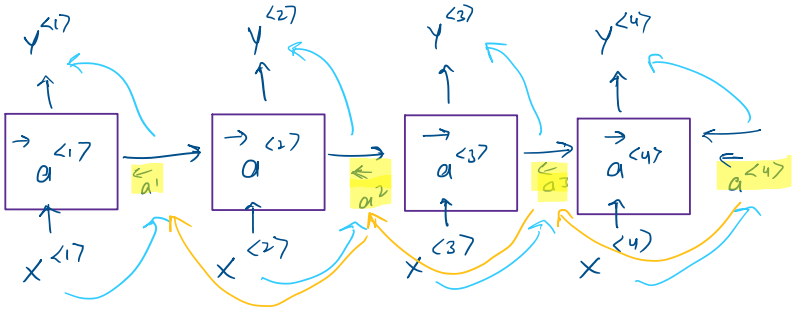


He said "Teddy Bears are on sale"

He said "Teddy Roosevelt was a great President"

- Unidirectional RNN.

$$\hat{y}^{<t>} = g\left(W_y\left[\vec{a}^{<t>}, \overleftarrow{a}^{<t>}\right] + b_y\right)$$

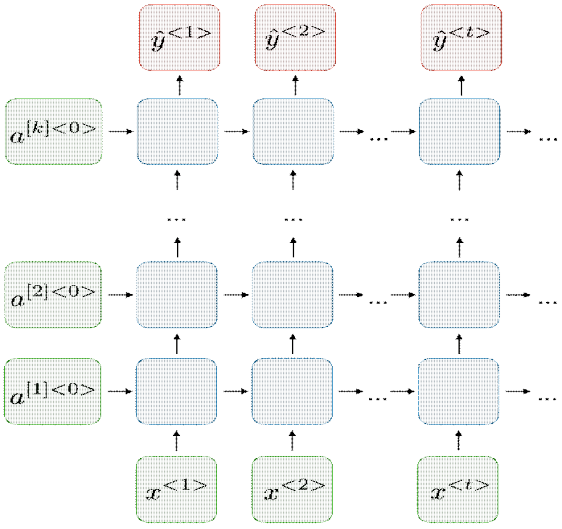


Acyclic graph

4 word sentence.

"BRNN with LSTM"

Deep RNN.



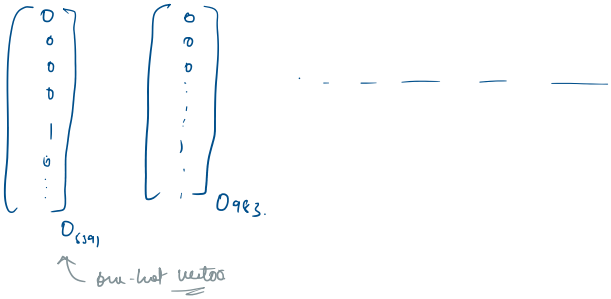
M1 Introduction To Word-Embeddings

Word Representation

$V = [a, aaron, \ldots\ldots Zulu, <UNK>]$

1 hot representation

Man	Women	King	Queen
(5391)	(9853)	(4914)	(7157)



Featured Representation

	Man	Women	King	Queen	Apple	Orange
gender	-1	1	-0.95	0.97	0.00	0.01
royalty	.01	.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	.03	-0.2
food						
Size						
⋮						

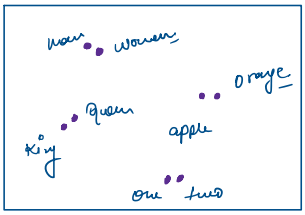
e_{5391}

"I want a glass of Orange juice"

"I want a glass of apple juice"

* Orange and apple are highly correlated ("fruit").

Visualizing Word Embeddings

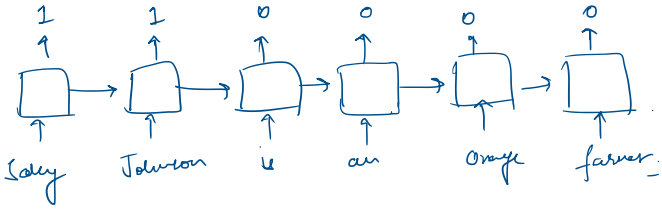


t- SNE.

USING WORD EMBEDDINGS

"Name Entity Recognition"

Sally Johnson is an Orange farmer.



"Robert is an durian farmer"
= a
Cultivator.

⇒ "Cultivator & farmer are highly Correlated."

Transfer learning

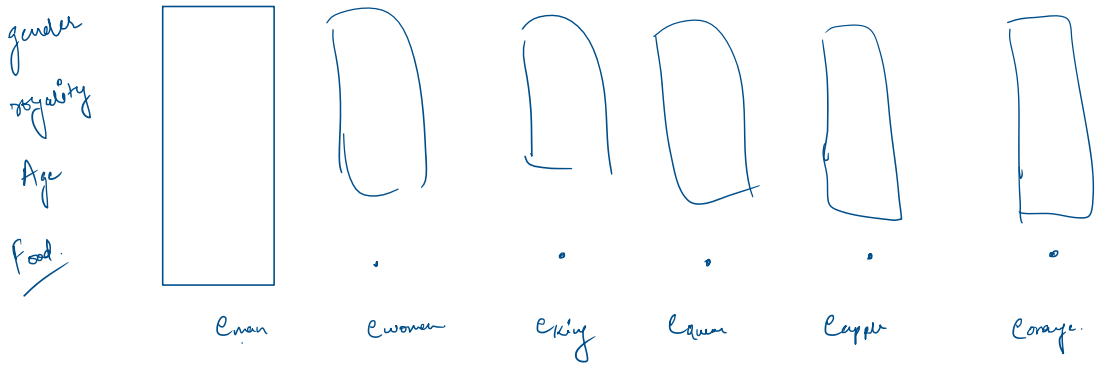
* > Learn word embedding from large text corpus.
* > Transfer embeddings to new task with smaller training set.
> Continue to fine tune the word embeddings with new data.

PROPERTIES OF WORD-EMBEDDINGS

Analogies.

Man : Women :: King : ?

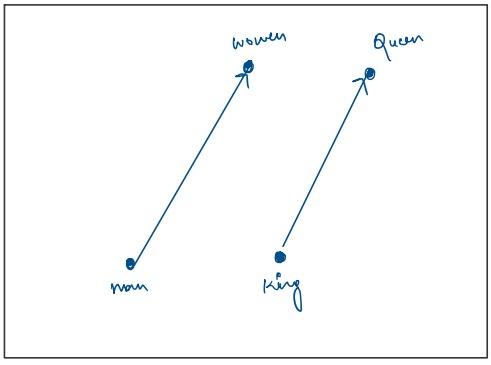
Man	Women	King	Queen	Apple	Orange
(5391)	(9853)	(4914)	(7157)	(456)	(6257)



$$e_{man} - e_{woman} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad e_{king} - e_{queen} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$e_{man} - e_{woman} \approx e_{king} - [\ldots]$$

$$\Rightarrow \boxed{e_{man} - e_{woman} \approx e_{king} - e_{queen}}$$



word embeddings

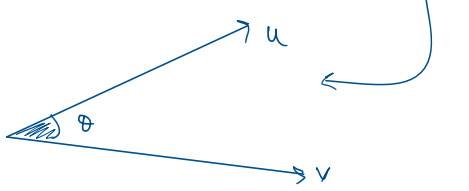
find word w, s.t.:

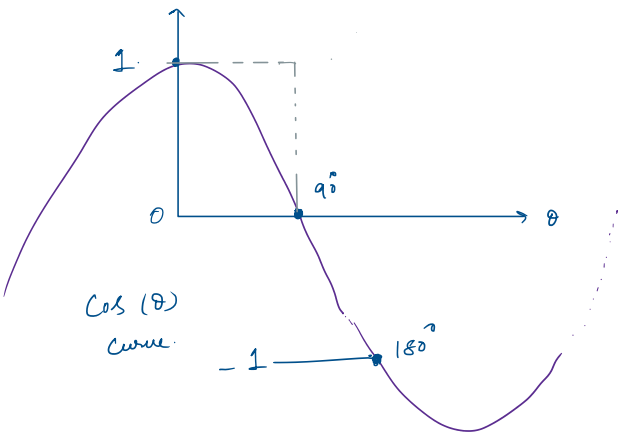
$$\max \text{Sim.}(e_w, e_{king} - e_{man} + e_{woman})$$

- find a similarity fx.

$\Rightarrow$ Cosine similarity.

$$\text{sim}(u, v) = \frac{u^T v}{\|u\| \|v\|} = \cos(\theta)$$





θ	Cos(θ)
0	1
π/2	0
π	-1
2π	1

Other similarity fx:

- Euclidian distance. $\|u - v\|^2$

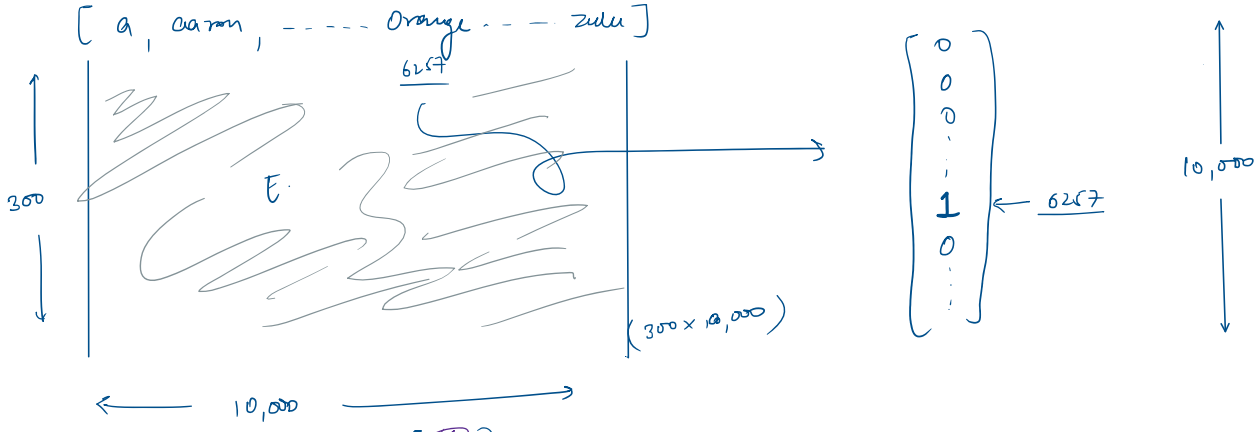
India : delhi :: Kenya : "Nairobi"

EMBEDDING MATRIX.

embedding matrix (E)

(300 x 10,000) → if you have 10,000 word dictionary

[a, aaron, ... Orange ... zulu]



$E \cdot O_{6257} = e_{6257}$

O_{6257} → one-hot vector.

$E \cdot O_j = e_j$

Embedding for word j.

↳ Embedding for words.

* It is not efficient to use multiplication of matrix, as the cost is very high.

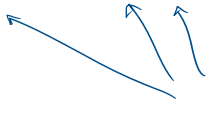
M2 LEARNING WORD EMBEDDINGS : Word2Vec and Glove.

LEARNING WORD EMBEDDING

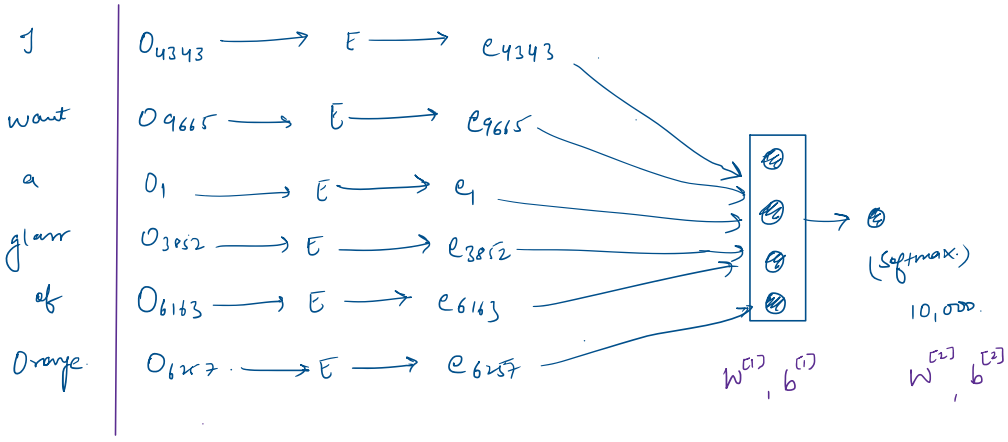
Natural language Model.

eg I want a glass of Orange _____.

(4343) (9665) (1) (3852) (6163) (6257)



index in vocab.



WORD 2 VEC.

Skip-grams : I want a glass of orange juice along with my cereal.

Context	Target
Orange	juice
Orange	glass
Orange	my

randomly selected

Model

Vocab - Size = 10,000 K

Context c ("Orange") (6257) → Target t ("juice") (4834)

$x \longrightarrow y$

$O_c \rightarrow E \rightarrow e_c \rightarrow$ [Softmax unit] $\rightarrow \hat{y}$

$e_c = E \cdot O_c$

Softmax : $p(t/c) = \dfrac{e^{\theta_t^T \cdot e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T \cdot e_c}}$

θ_t = parameter associated with o/p t (chance of t, being a label).

$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{1000} \left[y_i \cdot \log \hat{y}_i \right]$

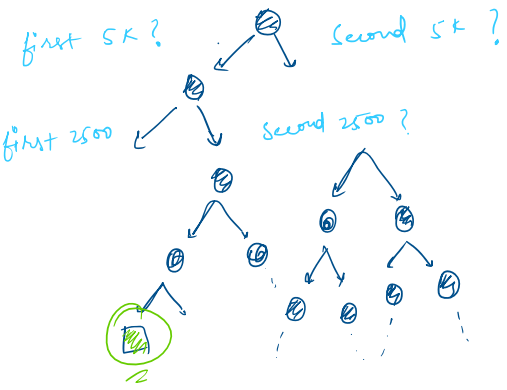
Problems with Softmax Classification.

- Computational Speed.

$p(t/c) = \dfrac{e^{\theta_t^T \cdot e_c}}{\sum_{i=1}^{10,000} e^{\theta_j^T \cdot e_c}}$

every time it will calculate 10K times, at higher values, it will compute slower.

⇒ Hierarchical Softmax.



NEGATIVE SAMPLING

predict (context, target) pair:

Context	target	Probability
Orange	juice	≈ 1
Orange	king	≈ 0
Orange	book	≈ 0

K

find p with random K target and fill from (0).

" Create a supervised learning algorithm, given (context, target) predict probability. "

$$K = \begin{cases} 5-20, & \text{for small datasets} \\ 2-5, & \text{for larger datasets} \end{cases}$$

Softmax Model :>

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{i=1}^{10,000} e^{\theta_j^T e_c}}$$

Context	word	target
- - - -	- - -	1
- - - -	- - -	0
- - -	- - -	0
- - -	- -	0

use logistic Regression Model.

$$p(y=1 \mid c,t) = \sigma(\theta_t^T e_c)$$

→ Sigmoid

" Orange "
6257

$O_{6257} \rightarrow E \rightarrow e_{6257}$ → juice? king? ⋮

1, real ex.
k, random ex.

Selecting -ve examples?

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{i=1}^{10,000} f(w_j)^{3/4}} = \frac{1}{|V|}$$

uniform distribution raised to 3/4 th.

$\sum_{i=1}^{10,000} f(w_{ij})^{3/4}$ $|V|$ to 5/9 ??.

GloVe Word Vectors

global vectors for word representation.

" I want a glass of Orange juice to go along with my cereal "

X_{ij} = # of times j appears in the context of i
(j = t, i = c)

minimize $\sum_{i=1}^{10K} \sum_{j=1}^{10K} f(X_{ij}) \left(\theta_i^T e_j + b_i + b_j - \log X_{ij} \right)^2$

$i, j = t, c$

or $(\theta_t)^T . e_c$

correlation

$f(X_{ij})$

this is the f at ------

(stop words)

* might give more weight to stop words and less on more attention words.

if $(X_{ij} = 0)$ then $\log 0$ = undefined, so we will add weighting term $f(X_{ij})$, $f(X_{ij}) = 0$, if $X_{ij} = 0$.
so $0 . \log 0 = 0$

f can be chosen to be heuristic to accomplish this.

Take average after calculating.

$$e_w^{(final)} = \frac{e_w + \theta_w}{2}$$

M3.

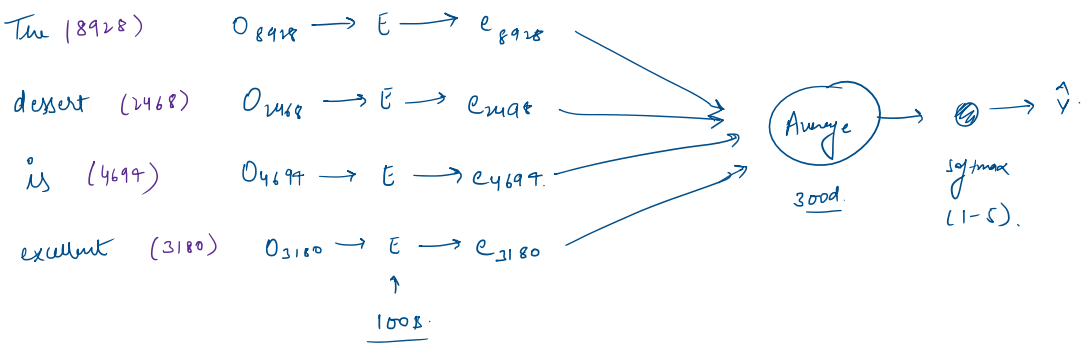
Applications Using Word Embeddings.

Sentiment Classification.

x ⟶ y

" dessert is excellent " ★★★★★

"dessert is excellent"	5 ★
"Service is Slow"	1 ★
"good for quick meal, but nothing special"	3 ★



> ignores word order.

ex: "Completely lacking in good taste, good service and good ambience."

* negative review, but "good" appears a lot!

RNN for Sentiment Classification

Completely lacking ---------- ambience.

O_{6538} for all

E_{6538}

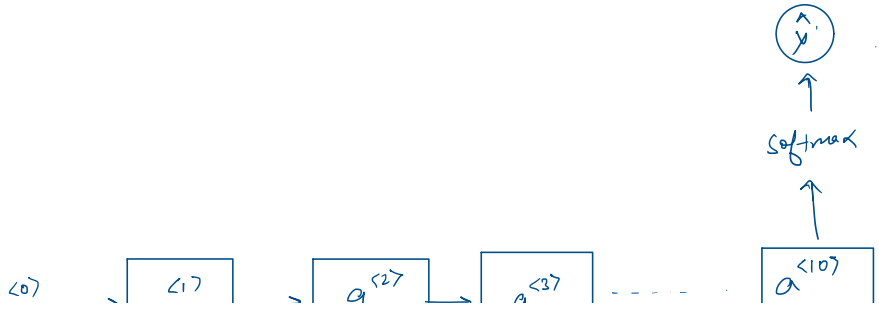
e_{6538}

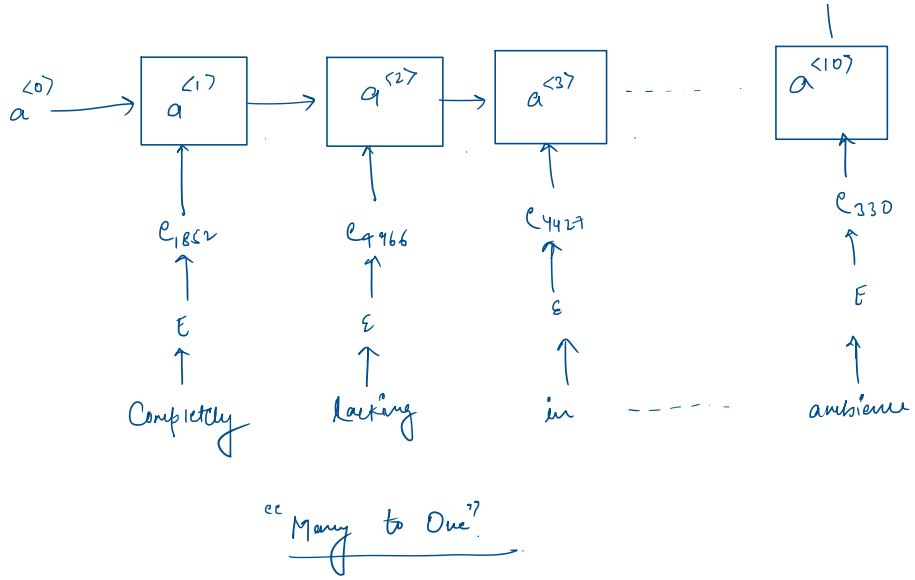
① Calculate one-hot vector.

② Embedding Matrix

③ find embedding vector. ($O * E$).

④ Feed e to RNN





"Many to One"

DEBIASING WORD EMBEDDINGS

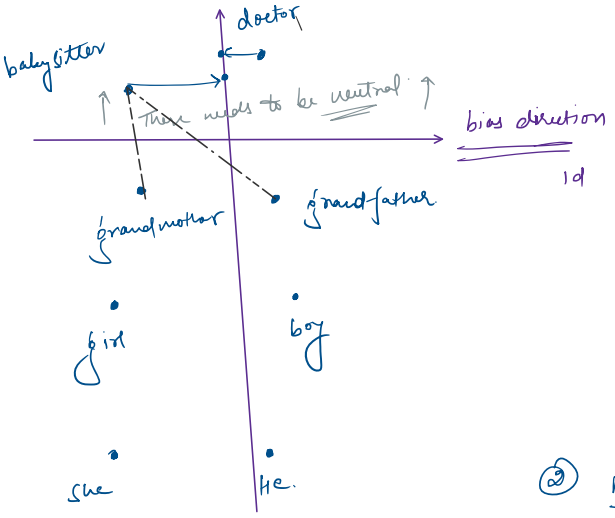
Man : Women :: King : Queen

Man : programmer :: Women : Homemaker ✗

Father : doctor :: Mother : Nurse ✗

* Word embeddings can reflect gender, ethnicity, age, sexual Orientation, and other biases of text used to train the model.

non-bias direction 299d



① Identify bias direction

(Taking gender bias here.)

$e_{he} - e_{she}$

$e_{male} - e_{female}$

$e_{boy} - e_{girl}$

} Average

② Neutralize: For every word that is not definitional, project to get rid of bias.

③ Equalize pairs :-

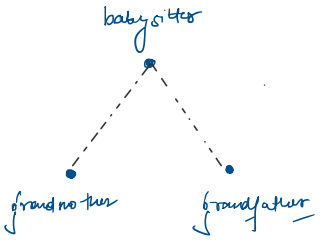
grandmother	grandfather
girl	boy

} should be at equivalent distance.

grandmother | grandfather } should be at equivalent distance.

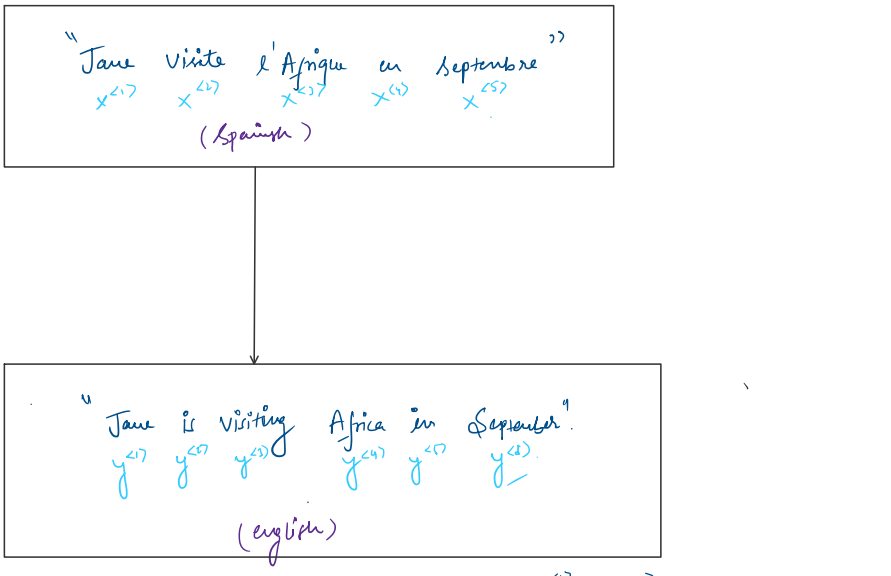
girl | boy

* Move grandmother and grandfather at pair points equidistant from a axis.



M1. Various Sequence to Sequence Architectures.

Basic Model



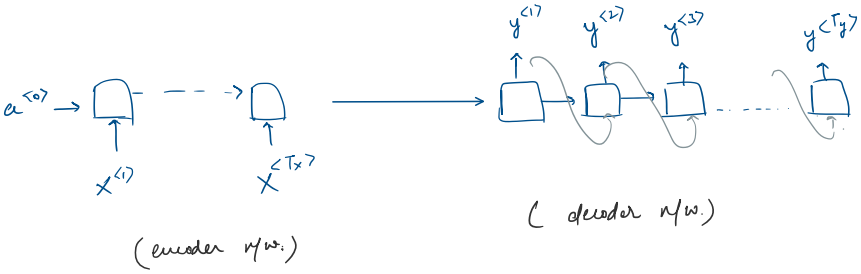
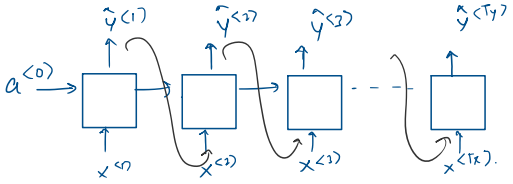


Image Captioning

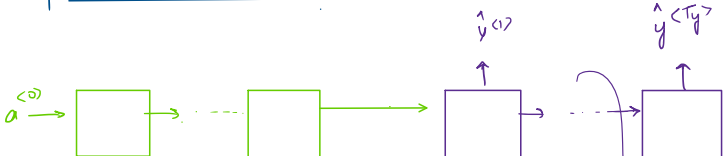
→ "A cat sitting on a chair."

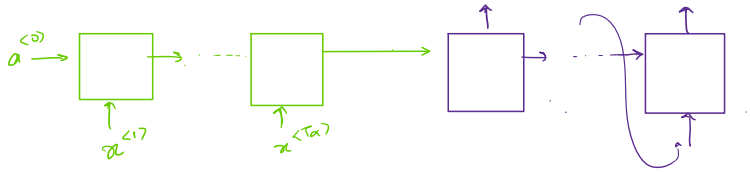
Picking the most likely sentence

Language Model (as in first week)



Machine Translation Model ⇒





encoded n/w. decoded n/w.

Conditional N/W.

" Jane visited l' Afrique en Septembre."

$P(\hat{y}^{<1>}, \hat{y}^{<2>}, \hat{y}^{<3>}, \hat{y}^{<4>} \ldots\ldots)$

→ Jane is visiting Africa in September. — good translation

→ Jane is going to be visiting Africa in September.

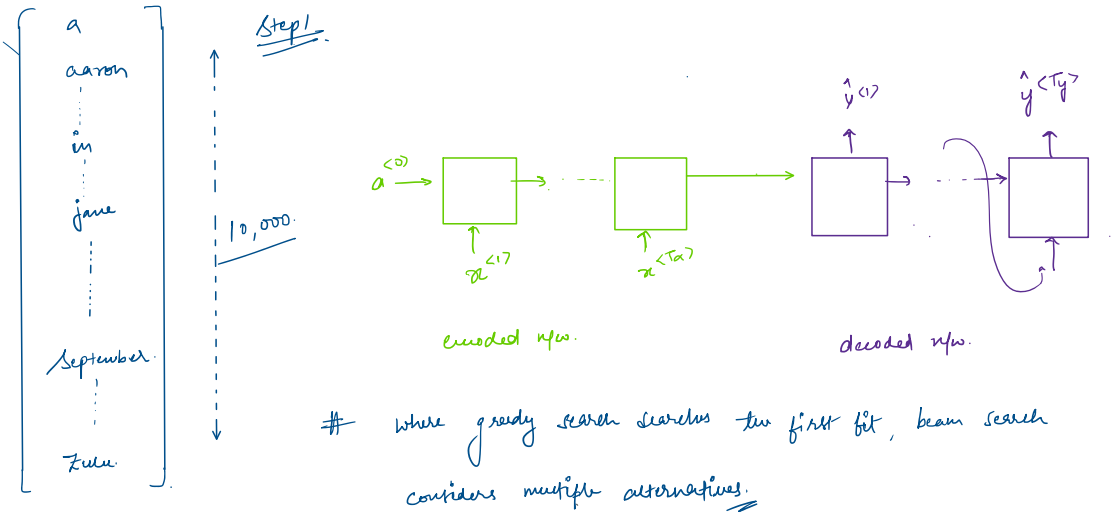
→ In September Jane will visit Africa.

→ Her African friend welcomed Jane in September. — bad translation

" beam search"

Beam Search.

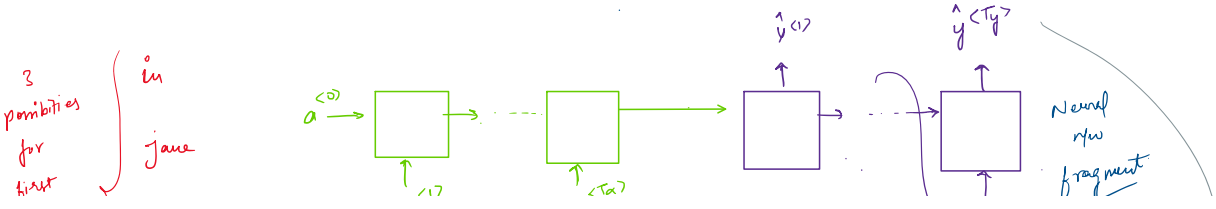
> Beam Search Algorithm

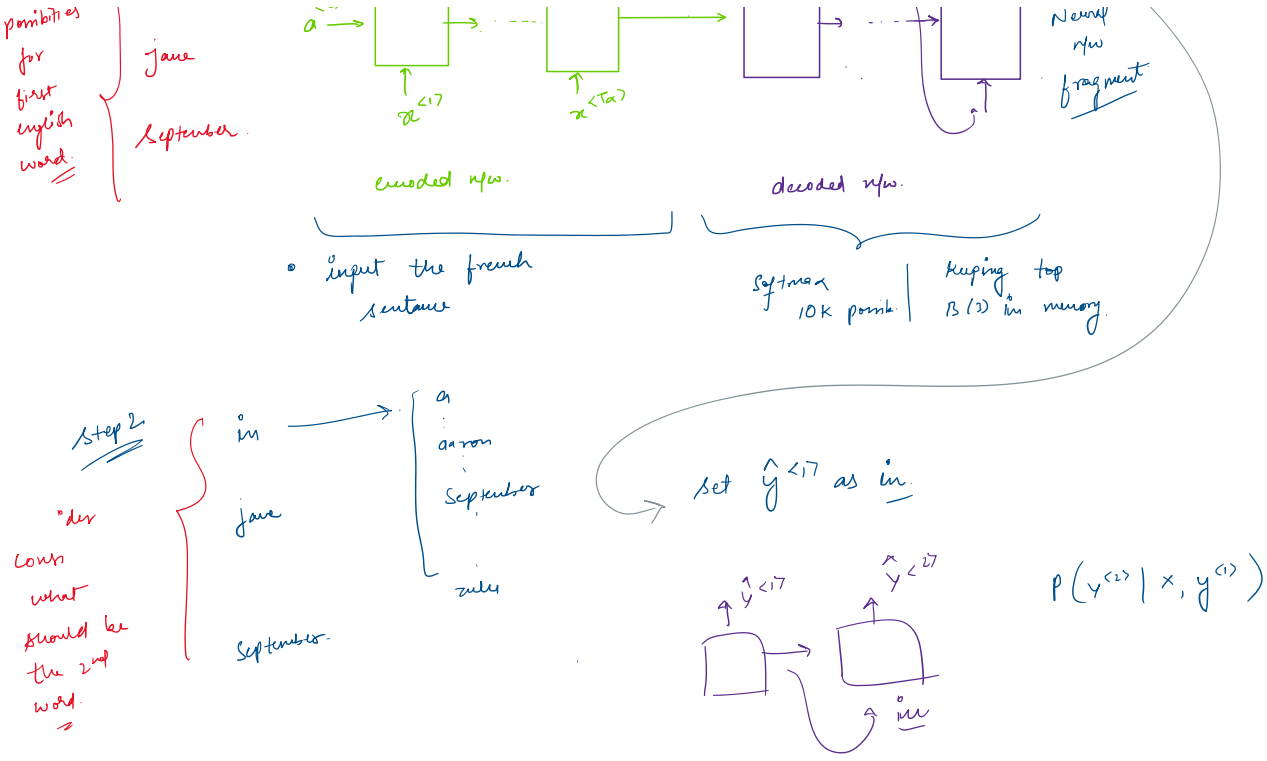


where greedy search searches the first bit, beam search considers multiple alternatives.

* beam search algorithm has a parameter B (Beam width)

B = 3 (say). • Considers 3 possibilities.





pairs of $y^{<1>}, y^{<2>}$ given X.

$$P(\hat{y}^{<1>}, y^{<2>} | x) = P(y^{<1>} | x) \cdot P(y^{<2>} | x, y^{<1>})$$

Conditional probability.

... for all the words.

Refinements to Beam Search

Length Normalization

$$\arg\max_{y} \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \ldots, y^{<t-1>})$$

log

$$P(y^{<1>} \ldots y^{<T_y>} | x) = P(y^{<1>} | x) \cdot P(y^{<2>} | x, y^{<1>}) \cdot P(y^{<3>} | x, y^{<1>}, y^{<2>})$$

* Normalize by number of words.

$$\frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \ldots, y^{<t-1>})$$

Beam Width ?

Larger :- Higher Accuracy, more Computational.

Smaller :- Low Accuracy, Faster

Error Analysis On Beam Search.

Beam search is an approximate search algorithm / heuristic search algorithm.

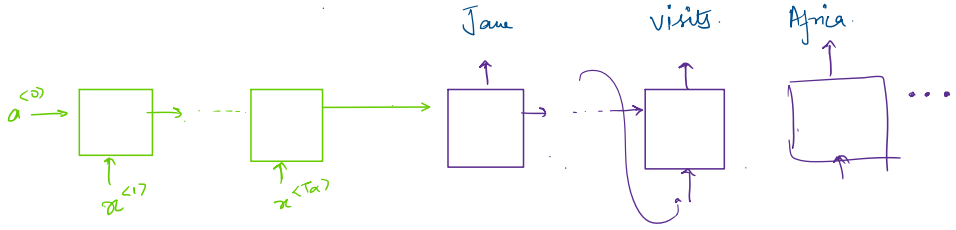
"Jane visite l'Afrique en septembre."

Human : Jane visits Africa in September. (y^*)

Algorithm : Jane visited Africa last September. $(\hat{y})$

- RNN (encoder and decoder)
- Beam Search.

RNN.



max. $\left(P(y^*|x) \;,\; P(\hat{y}|x) \right)$

Jane visits Africa in September. (y^*) $P(y^*|x)$

Jane visited Africa last September. $(\hat{y})$ $P(\hat{y}|x)$

Case 1 : $P(y^*) > P(\hat{y})$

⇒ but beam search chooses $P(\hat{y})$.

⇒ Beam Search is at fault.

Case 2 : $P(y^*) < P(\hat{y})$

⇒ y^* is a better translation than $\hat{y}$. But RNN predicts

$$P(y^*) < P(\hat{y})$$

⇒ RNN model is at fault.

Bleau Score.

(Bilingual Evaluation understudy Score.)

- given a french sentence that can be multiple english translation, this can be solved with "Bleau Score"

French : Le chat est sur le tapis.

Reference 1 : The cat is on the mat. { the : 2 }

Reference 2 : There is cat on the mat. { the : 1 }

BLEAU Score measure how good a machine translation is.

Precision (1) := $\frac{2}{7}$

Blue Score on Bigrams

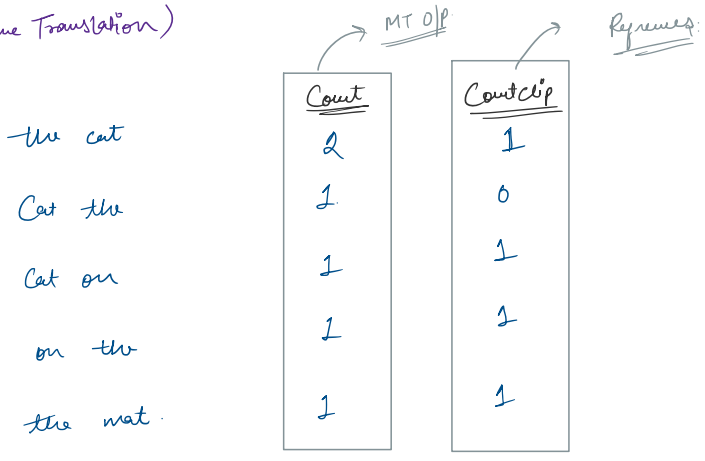
"pairs of words next to each other"

Reference 1 : The cat is on the mat.

Reference 2 : There is cat on the mat.

MT O/P : The cat the cat on the mat

(Machine Translation)



Modified bigram Precision :- $\frac{\sum \text{Count clips}}{\sum \text{Count}}$

$$= \frac{4}{6} = \frac{2}{3}$$

Bleau Details

p_n = Bleau Score of n-grams.

p_1, p_2, p_3, p_4

Combined blue Score : $BP \exp\left(\frac{1}{4}\sum_{n=1}^{4} p_n\right)$.

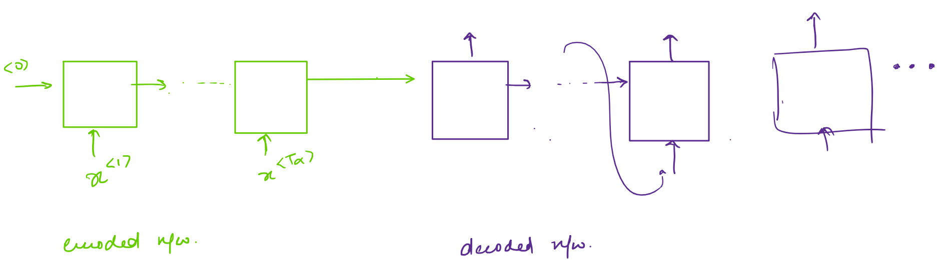
(brevity penalty)

$$BP \begin{cases} 1, & \text{if MT\_Output length} > \text{reference o/p Length} \\ \exp(1 - \text{reference\_o/p Length} / \text{MT\_o/p Length}) \end{cases}$$

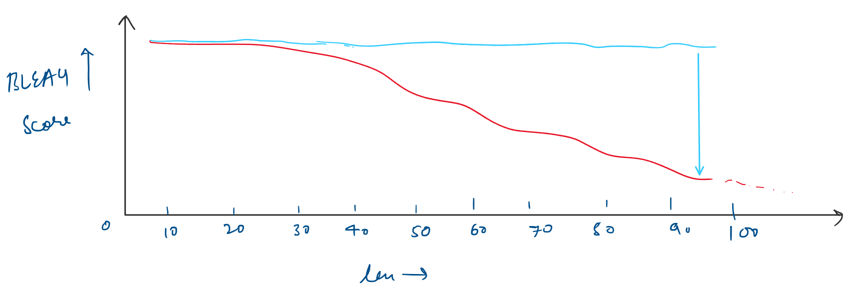
Attention Model Intuition

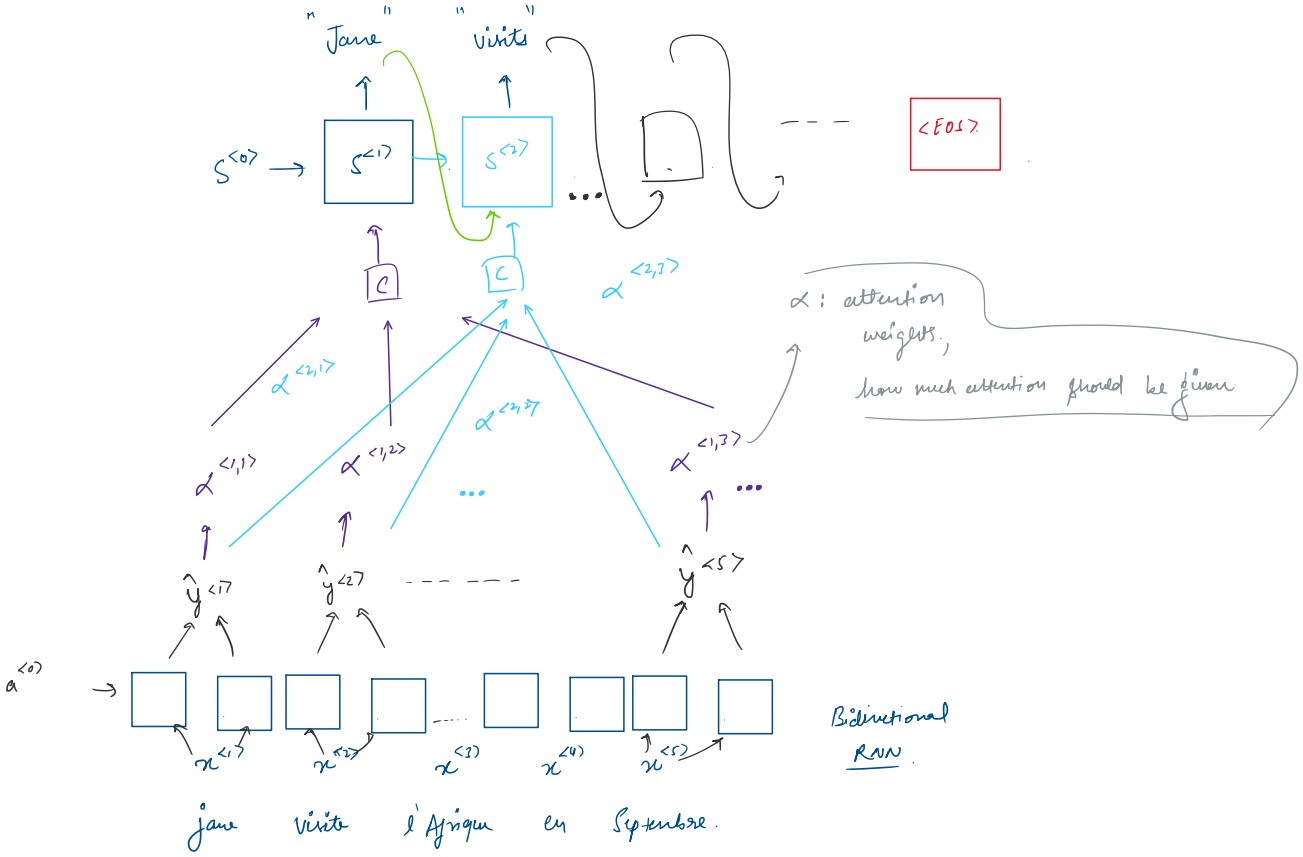
The problem of long Sequence

> given a long french sentance ______________

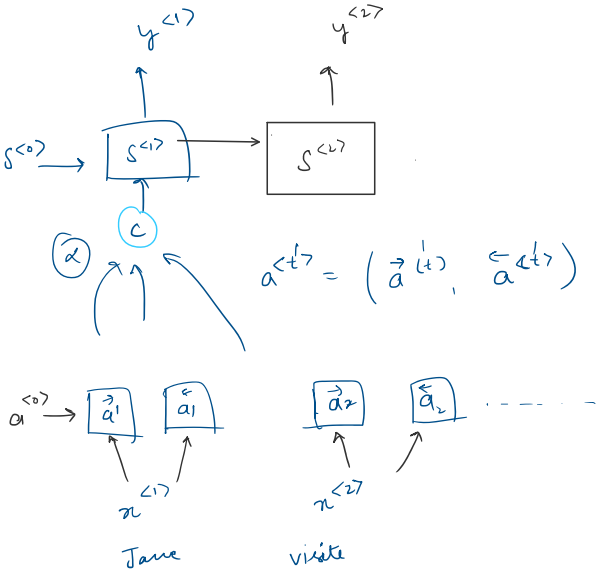


* It is hard to memorize the whole sentance, what humans will do is translate a part of sentance and move forward.





Attention Model



$$\sum_{t'} \alpha^{<1,t'>} = 1$$

$$c^{<1>} = \sum_{t'} \alpha^{<1,t'>} a^{<t'>}$$

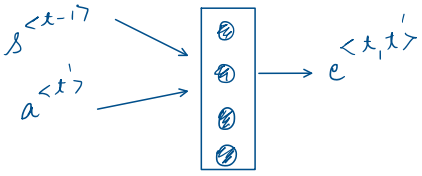
$\alpha^{<t,t'>}$ = amount of attention $y^{<t>}$ should pay to $a^{<t'>}$

Computing Attention

$a^{<t,t'>}$ = amount of attention $y^{<t>}$ should pay to $a^{<t'>}$

$$\frac{\exp(e^{<t,t'>})}{\sum_{t'} \exp(e^{<t,t'>})}$$

$$\alpha^{\langle t,t' \rangle} = \frac{\exp\left(e^{\langle t,t' \rangle}\right)}{\sum_{t'=1}^{T_x} \exp\left(e^{\langle t,t' \rangle}\right)}$$



- takes quadratic cost. $(T_x)^{T_y}$

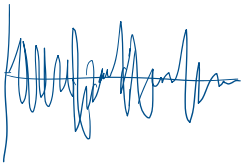
T_x = input
T_y = Output

M2 Speech Recognition

Speech recognition

$x \longrightarrow y$

(audio clip) (transcript)

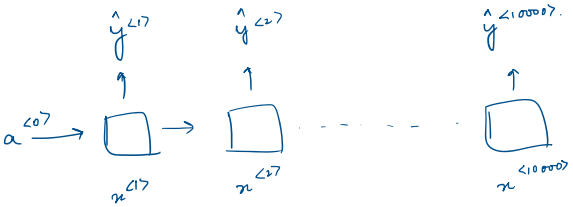


time →

"the quick brown fox ---"

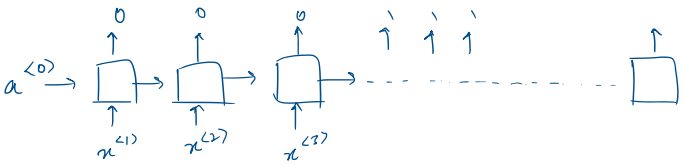
CTC Cost

"Connectionist temporal classification"



Trigger Word detection

Spectogram



"Hey Siri !!"

"OK google !!"